

Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

Projets

Note : Le projet donnera lieu à une soutenance avec démonstration du programme écrit en Java, et à un rapport **impérativement** rédigé en LaTeX.

Les notions abordées dans chacun des sujets n'ont pas forcément été vues à votre niveau, mais les sujets sont suffisamment explicites pour que vous puissiez faire le projet de manière intuitive.

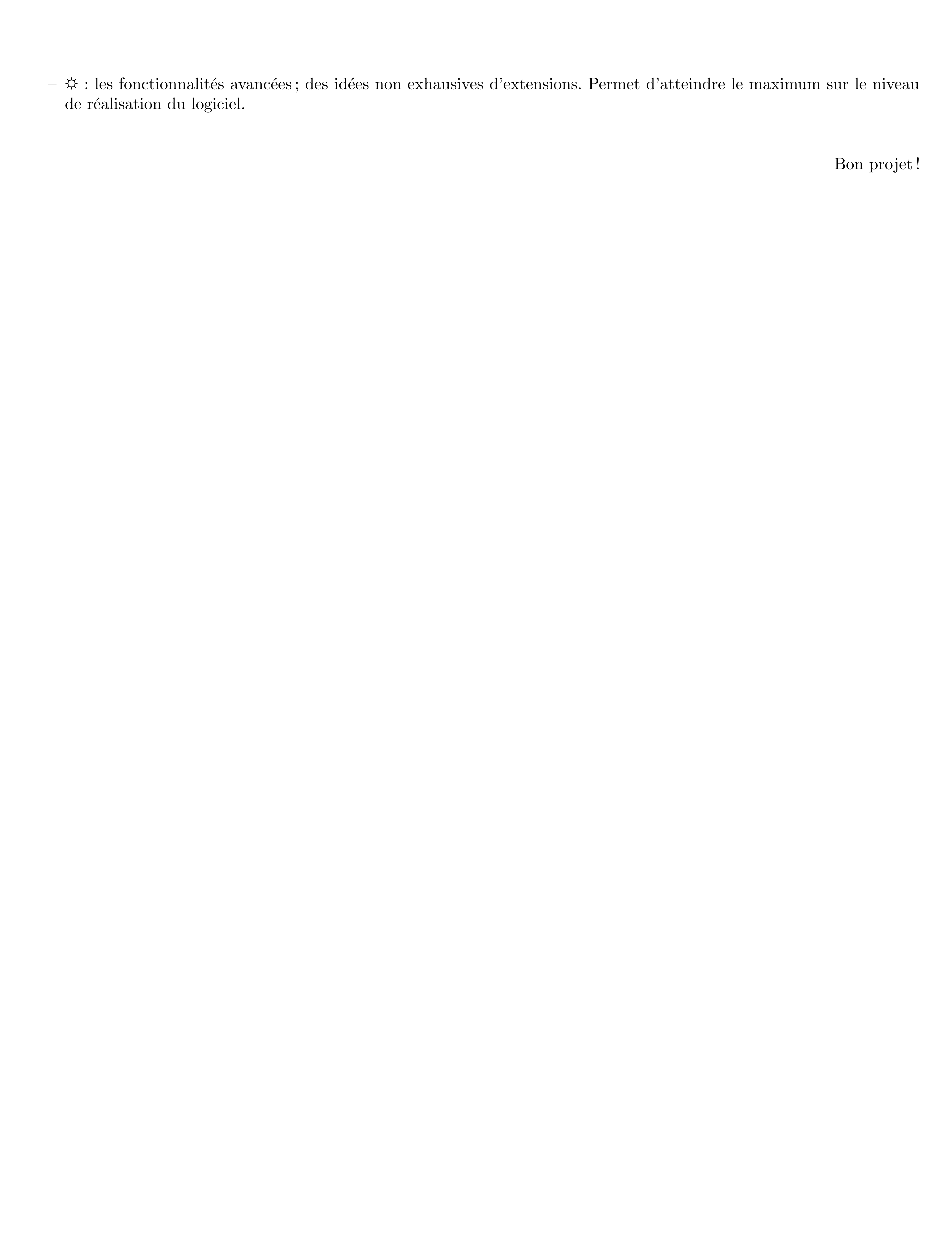
Le projet devrait ainsi vous permettre de vous familiariser (voire de vous documenter) avec des notions que vous aborderez plus tard dans votre cursus et de saisir ainsi la problématique et les applications induites par ce sujet.

La notation du projet se décomposera en :

1. l'assiduité, le rythme d'évolution du projet et l'organisation des tâches
   - la présence et la participation
   - les points d'avancement
2. le logiciel proprement dit
   - le niveau de la réalisation
   - la conception du logiciel
   - la qualité du code : performance des algorithmes, robustesse du programme
   - la propreté du code : respect des conventions de codages et les commentaires.
   - la modularité du code : modèle-vue-contrôleur, extensions facilement "intégrables", la portabilité du code
   - la javadoc
   - la batterie de tests
3. le pré-rapport et le rapport
   - l'analyse du sujet
   - la description de votre logiciel : l'architecture globale, le diagramme UML de classes, les algorithmes
   - le manuel utilisateur
   - les remarques pertinentes et perspectives d'évolution
   - la forme : la structure, le style, la grammaire et l'orthographe
   - l'utilisation de LaTeX
4. la soutenance
   - la qualité des supports de soutenances (transparents)
   - le discours (expression orale, clarté de la voix, regard, posture)
   - les réponses aux questions
   - la forme : le plan, la structure, le style, l'orthographe, la lisibilité
   - le respect du temps imparti

Chaque prototype devra comporter deux types d'exécution : une exécution (plus pour du batch ou du débogage) en mode console et une exécution avec une IHM graphique ergonomique.

Pour chaque sujet, plusieurs fonctionnalités sont demandées. Ces fonctionnalités sont classés en trois catégories :
- ◖ : les fonctionnalités de base ; le minimum pour atteindre tout juste la moyenne sur le niveau de réalisation du logiciel ;
- ○ : les fonctionnalités attendues ; permet d'atteindre les 3/4 du niveau de réalisation du logiciel ;



- ☼ : les fonctionnalités avancées ; des idées non exhaustives d'extensions. Permet d'atteindre le maximum sur le niveau de réalisation du logiciel.

Bon projet !

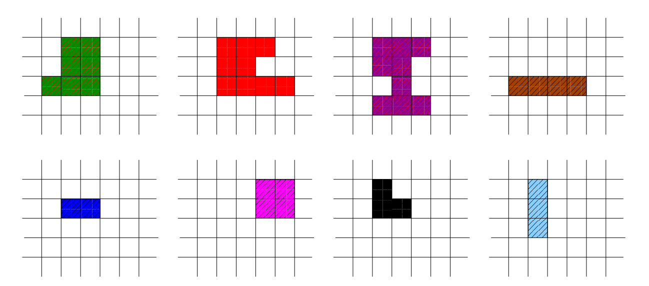
Table des matières



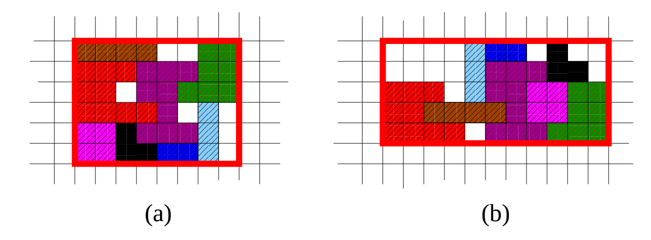
Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

1- Agencement de formes de manière optimale**

Soit des formes aléatoires représentées sous forme de cases **connexes** sur un quadrillage ainsi que figurées ci-dessous.



Le but du jeu est de pouvoir à partir d'un ensemble de formes données au départ de les agencer afin de former le rectangle englobant possédant le moins de cases inutilisées. Deux exemples d'agencement des formes données ci-dessus sont représentées sur les figures (a) et (b) ci-dessous. Le rectangle englobant est le rectangle minimal contenant toutes les formes ainsi agencées. Il est représenté en trait fort rouge sur les schémas.



(a) (b)

Dans le cas (a), il reste 7 cases inutilisées dans le rectangle englobant. Dans le cas (b), il reste 14 cases inutilisées dans le rectangle englobant.

Le placement des formes dans le cas (a) est donc plus optimal que dans le cas (b).

Le programme demandé est de :

1. ◖ Générer n formes aléatoires contenant chacune au plus m cases connexes. n et m devront être paramétrables par l'utilisateur.
2. ◖ Proposer à l'utilisateur de les placer. Calculer ensuite le rectangle englobant et le nombre de cases vides résultants.
3. ○Votre logiciel devra également être capable de proposer une solution optimale au problème.
4. ☼Vous pourrez par la suite, enrichir le jeu en permettant la rotation des formes lors du placement.

Mots-clefs :



Licence 2 I

2010–2011

Génie logiciel

T.T. Dang Ngoc
dntt@u-cergy.fr

2- Agent réactif simple et rationnel**

Soit une grille de $N \times N$ cases sur laquelle évolue un gardien. Des obstacles sont disposés aléatoirement sur le terrain suivant des densités données :

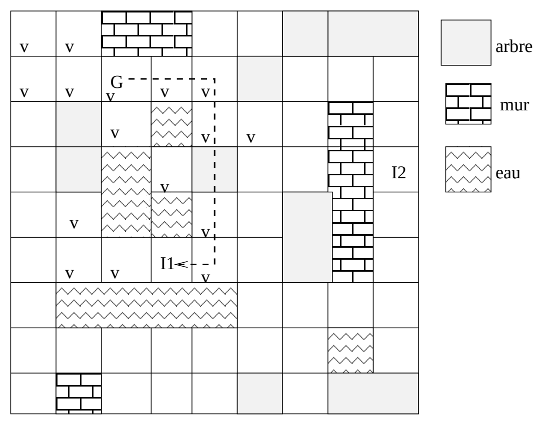
– certains obstacles « arbre » empêchent le gardien de voir au-delà de cet obstacle mais ne l'empêchent pas de passer ;
– d'autres obstacles « eau » empêchent le gardien de passer sur cette case mais ne l'empêchent pas de voir au delà ;
– enfin les obstacles « mur » empêchent le gardien à la fois de voir et de passer.

Lorsqu'un intrus se trouve dans son champ de vision, le gardien se dirige vers lui pour l'attraper (*ie.* se mettre sur la même case) par le chemin le plus court (en tenant compte bien sûr des obstacles infranchissables). Bien que le gardien ait un champ de vision limité, on supposera qu'il connaît la carte (types d'obstacles compris) par cœur pour pouvoir élaborer son chemin. Bien sûr, le gardien ne connaît pas la position des intrus sans les avoir vus.

Pendant son parcours, si le gardien voit d'autres intrus, il le note et les attrapera après s'être occupé des intrus qu'il est en train de traiter. Une fois un intrus repéré, le gardien est capable d'établir un chemin vers lui même si celui-ci est en dehors de sa vision.

Lorsque le gardien n'a aucun intrus dans son champ de vision, il patrouille au hasard (déplacement sur des cases contigües valides).

La figure ci-dessous donne un exemple de déplacement d'un gardien.



v Cases visibles par le gardien

Le but du programme à réaliser est de :

1. ◖ Initialiser une grille :
   – soit de manière aléatoire suivant certains paramètres donnés par l'utilisateur (densités des obstacles, nombre d'intrus, etc.)
   – soit de manière manuelle par l'utilisateur.
2. ◖ Exécuter pas à pas les actions du gardien.
3. ◖ Les intrus seront tout d'abord statiques (ils ne bougent pas).
4. ◖ L'utilisateur peut prendre controle du gardien pour jouer lui-même à attraper les intrus en un minimum de coups
5. ○Les intrus seront dynamiques (ils se déplacent eux-même au fur et à mesure du déplacement du gardien). Dans un premier temps, les intrus bougent de façon aléatoire, dans un second temps, les intrus se déplaceront **en fonction** de la position du gardien (ils essayeront de s'en éloigner le plus possible).
6. ☼simuler les actions de g gardiens et i intrus sur une même grille. Les gardiens pouvant se coordonner pour attraper les intrus en un minimum de temps.

Mots-clefs : Agents



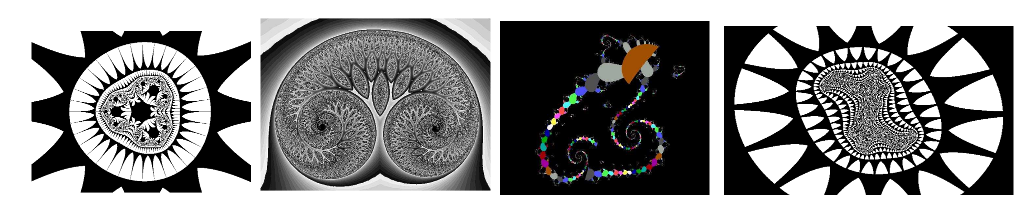
Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

3- Biomorphes**

De nombreuses formes naturelles peuvent se représenter sous forme de fonctions mathématiques.

Pour construire un biomorphe, on considère un réseau de points dans un rectangle du plan complexe : les coordonnées de chaque point du réseau constituent les parties réelles et imaginaires de diverses valeurs initiales z_0. A chaque point du réseau, on associe d'autre part un pixel. Selon la valeur des parties réelles ou imaginaires obtenues après itérations de la fonction, on fait varier les couleurs du point correspondant.

Voici quelques exemples de biomorphes générés :



On se place dans le plan complexe formé des points d'affixe z = x + iy. On considère une suite complexe définie par :

$$u_0 = z_0 u_{n+1} = f(u_n)$$

où f est une fonction continue complexe ayant un point fixe. Le nombre complexe z est composé de deux parties, l'une dite réelle et l'autre imaginaire, s'écrivant sous la forme $z = a + ib$. Dans le plan complexe, z désigne l'affixe d'un point où la partie réelle a en détermine l'abscisse, et la partie imaginaire b, l'ordonnée.

c représente les coordonnées du point du plan en cours de calcul.

Chaque biomorphe sera contenu dans un carré délimité du plan général.

- ◖ Dans un premier temps vous représenterez des biomorphes en permettant à l'utilisateur de définir son équation
- ○Vous permettrez à l'utilisateur d'ajuster la représentation de chaque biomorphe par :
  - rotation
  - changement d'échelle
  - translation
  - colorisation
- ○Vous permettrez à l'utilisateur de définir une trajectoire à chacun de ses biomorphes. Les opérations précédentes évoluant au cours de la trajectoire suivant des équations ou paramétrage bien définis par l'utilisateur.
- ☼deux biomorphes se rencontrant au cours de leur trajectoire respectives peuvent donner naissance à un autre biomorphe dont les paramètres, l'équation de forme et l'équation de trajectoire seront issus (suivant des critères à définir) des paramêtres des deux biomorphes parents.

Références :

`http://utbiom.free.fr/Documentation/Biomorphes_Article_Pour_la_Science.pdf`
`http://mathenjeans.free.fr/amej/edition/actes/actespdf/91091093.pdf`
`http://www.madteddy.com/biomorph.htm`

Mots-clefs : biomorphe, fractale, géométrie



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

4- Simulateur d'élements physiques**

L'assemblage d'éléments de la figure ci-dessous forme une **machine** permettant de réaliser la séquence d'action suivante : Lorsqu'on appuie sur le bouton de la lampe torche (1), le faisceau lumineux est envoyé vers un miroir (2) qui réfléchit la lumière sur une parabole (3) qui concentre et dirige le faisceau vers une lentille (4) qui concentre le faisceau sur une corde (5) reliée à un poids (6) d'un côté et une balançoire (10) en passant par deux deux poulies (7) et (8). Le faisceau brûle la corde (5) qui fait tomber le poids (6) et relâche la tension sur la balançoire sur laquelle se trouve une balle (9) d'un côté et un poids (11) de l'autre. La balancoire ainsi libérée, et le poids (11) étant important, la balle (9) est expédiée suivant une trajectoire parabolique et finit sa course dans un entonnoir (12) relié à un tuyau (14) via un coude (13). À la sortie du tuyau, la balle tombe sur un ressort (15) qui fait monter la balle contre la cloche (16) qui sonne.

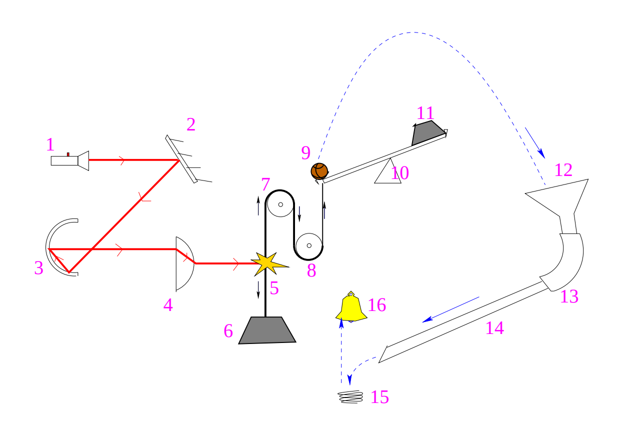


Fig. 1 – Exemple de machine

Il s'agit dans ce projet d'offrir à l'utilisateur un ensemble d'objets qu'il pourra disposer à sa guise afin de réaliser l'encaînement d'actions qu'il voudra. Ces objets utiliseront des phénomènes

- des objets utilisant des phénomènes optique : miroir, parabole, lentille convexe ou concave, etc.
- des objets utilisant des phénomènes mécanique : poulies, engrenages, balances, ressorts, etc.
- des objets soumis à des forces : tels que balles, poids, etc.
- des objets intermédiaires : cordes, pentes, entonnoir, tuyaux divers, etc.
- des objets farfelus : un chat qui lorsqu'il entend le son d'une cloche se met à courir en ligne droite, un hamster dans sa roue qui lorsqu'il est affolé, se met à courir, générant ainsi de l'électricité (une dynamo, quoi !), etc.
- des objets utilisant des phénomènes électriques : lampe-torche qui réagit à une pression mécanique (le bouton) pour émettre un signal lumineux, aimants, etc.
- des objets divers : cloches, arcs, etc.

Chaque objet devra être **fortement paramétrables** (choix de la position initiale, de l'orientation, masse, inclinaison, solidité, courbure de lentilles, etc.).

Le comportement des objets devront respecter les lois de la physique (loi de la gravitation, principe d'action-réaction, etc.)

Votre travail consistera donc :

1. À offrir à l'utilisateur un panel d'objets le plus large et paramètrable possible et permettre à cet utilisateur de les placer.
2. À simuler ensuite le système.

3. ○À permettre à l'utilisateur de définir de nouveaux objets
4. ○À permettre à l'utilisateur de définir de nouvelles lois physiques ou simplement modifier les paramètres de lois physiques existantes (par exemple, la constante universelle de gravitation)
5. ☼Utiliser des objets utilisant des phénomènes thermodynamiques ou des phénomènes de flux : écoulement d'eau, pression de gaz, etc. tout en repectant les principes de la thermodynamique et de mécanique des fluides.

Mots-clefs : simulation physique



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

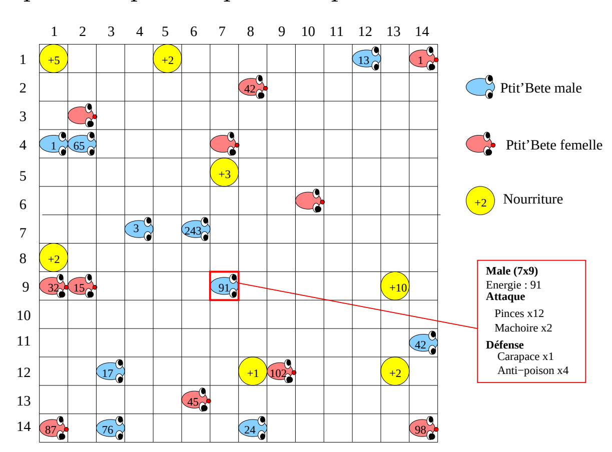
5- Mini-simulation d'une évolution génétique simplifiée**

Sur une grille de $N \times N$ cases évoluent des ptit'bêtes. La ptit'bête est un être primaire qui bouge, mange, vieillit et suivant son âge et son niveau d'énergie est capable de se battre, se reproduire.

Une ptit'bête est caractérisée par :

- un sexe : mâle ou femelle
- un ensemble de caractéristiques d'attaques (pinces, machoire, venin, etc.). Chaque attaque comporte un niveau d'efficacité (ex. pinces=0 veut dire que la ptit'bête n'a pas d'attaque "pinces", pinces=100 veut dire que la ptit'bête a une capacité d'attaque par pince de 100).
- un ensemble de caractéristiques de défense (carapace, épines, immunité, etc.). À chaque attaque correspond une ou plusieurs défense appropriées (ex. carapace contre pince, épines contre machoires, etc.). De la même manière que pour l'attaque, chaque défense comporte un niveau d'efficacité.
- un niveau d'énergie : mise à une valeur maximum déterminée à sa naissance, elle décroit en fonction des actions (déplacement, reproduction, combat) et s'accroit lorsque la ptit'bête mange ou gagne un combat. Elle ne peut en aucun cas dépasser la valeur maximum déterminée. Lorsque le niveau d'énergie atteint zéro, la ptit'bête meurt et disparaît de la carte.

Dans le reste de l'énoncé, on notera par exemple une ptit'bête par :



Chaque jour (un jour est représenté par un pas d'exécution), quelques ptit'bêtes se déplacent de zéro, une ou plusieurs cases. Plusieurs cas de figure se présentent.

- si la case sur laquelle elle tombe est vide, elle s'y rend et perd des points d'énergie ;
- si la case est occupée par de la nourriture, la ptit'bête mange et regagne un certain nombre de points d'énergie suivant la valeur nutritive de la nourriture ;
- si la case est occupée par une autre ptit'bête, alors deux cas sont possibles :
  - si l'autre ptit'bête est du même sexe, il y a combat, le vaincu perd un nombre important de points d'énergie (par exemple la moitié de ses points d'énergie maximum) et est placé à une case vide de la carte. Le vainqueur, lui, gagne un nombre important de points.
  - si l'autre ptit'bête est du sexe opposé, il y a reproduction si le niveau de points de vie de chacune des ptit'bêtes est supérieur aux trois-quarts de point de vie maximum. La ptit'bête résultante sera placée à une case vide de la carte.

Le combat de deux ptit'bêtes se fait par comparaison successive des caractéristiques attaques/défense des ptit'bêtes prises deux à deux en considérant à chaque fois les niveaux d'efficacité des attaques/défenses. La différence totale permet de désigner le vainqueur du combat. L'environnement permet d'influer sur les caractéristiques lors d'un combat par un coefficient multiplicateur (positif, nul ou négatif) donné pour chaque caractéristique. Un environnement pourra être défini de manière globale (toute la grille) ou partielle (certaines parties de la grille : par exemple, la montagne, le désert, la forêt, etc.).

La ptit'bête issue de la reproduction entre deux ptit'bêtes sera générée de la manière suivante :

- un sexe aléatoire
- un ensemble de caractéristiques : chaque caractéristique (et leur efficacité) sera récupérée soit de l'une ou de l'autre des ptit'bêtes (probabilité de 1/2), et dans un certain petit pourcentage m défini par l'environnement, il se peut que la caractéristique générée soit aléatoire et ne dépende d'aucun des parents (mutation).
- un niveau d'énergie maximum généré de la même façon que les caractéristiques
- le niveau d'énergie sera au départ initialisé au niveau d'énergie maximum.

Le programme demandé sera de :

1. ◖ Générer aléatoirement une grille de $N \times N$ cases et y placer M ptit'bêtes dont les caractéristiques seront tirées aléatoirement. Les paramètres seront déterminés par l'utilisateur.
2. ◖ Faire évoluer la grille jour par jour en y plaçant aléatoirement de la nourriture sur certaines cases vides suivant une densité d donnée.
3. ○L'environnement pourra être changé manuellement par l'utilisateur, de manière aléatoire ou périodique.
4. ☼Intégrer un système d'antennes, de vision ou d'odorat (dont la performance est paramêtrée génétiquement) permettant aux ptit'bêtes de repérer à n cases de là : de la nourriture, un partenaire idéal de reproduction, un rival potentiel, un adversaire plus fort que soi, etc.

Mots-clefs : Automate cellulaire



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

6- Convertisseur UML**

UML (en anglais Unified Modeling Language, « langage de modélisation unifié ») est un langage graphique de modélisation des données et des traitements.

Parmi tous les diagrammes définis dans la normalisation UML, nous nous intéresserons uniquement à :

- *Diagramme de classes :* il représente les classes intervenant dans le système. On y représente principalement
  - les classes avec leur nom, attributs et méthodes.
  - les relations entre classes : interface, héritage
  - l'accessibilité des attributs et méthodes
  - les classes contenues dans un paquetage.
- *Diagramme d'objets :* il sert à représenter les instances de classes (objets) utilisées dans le système.

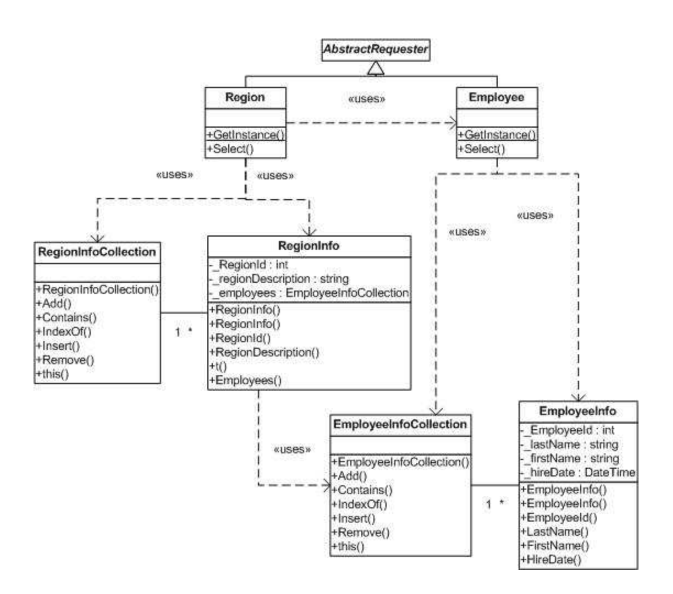


Fig. 2 – Exemple de diagramme de classe UML

Les explications sur le diagramme de classes peuvent être consultées aux l'URL suivantes :

`http://fr.wikipedia.org/wiki/Diagramme_de_classes`
`http://uml.free.fr/`

Dans un premier temps vous devrez permettre à l'utilisateur de dessiner son schéma UML en lui offrant les composants UML utilisés dans le diagramme des classes

- les paquetages : un paquetage peut être public ou privé.
- les interfaces, classes : une classe comporte un nom, des methodes et des attributs. Les méthodes et attributs peuvent être private, public, protected, ou "friend".
- instance
- les instances de classes.
- les relations d'héritage, d'implémentation d'interface, les lancements d'exception.

Vous prendrez garde que les types des méthodes, attributs et instance pouvant être d'autres classes, celles-ci devront être reconnues et pouvoir être mises en relation.

Dans un second temps, un générateur automatique devra pouvoir générer les différents fichiers java correspondant à votre fichier UML. Il s'appuiera autant que possible sur le diagramme d'objet pour la classe principale, et sur le diagramme de classes pour toutes les autres classes. Les importations nécessaires de paquetages devront être générées, ainsi que toutes les méthodes get/set, les déclarations de variables, les méthodes, les lancements d'exception, les relations d'héritage et implémentations d'interface.

Les commentaires javadoc devront être également inclus avec les entrées nécessaires (`@param, @return, etc.`). Les corps des méthodes et les commentaires seront évidemment vides et laissés à la discrétion du programmeur, mais devront néanmoins être compilables telles quelles (une méthode dont la signature n'est pas `void` devra retourner une valeur par défaut correspondant au type attendu).

Ecrivez le programme qui :

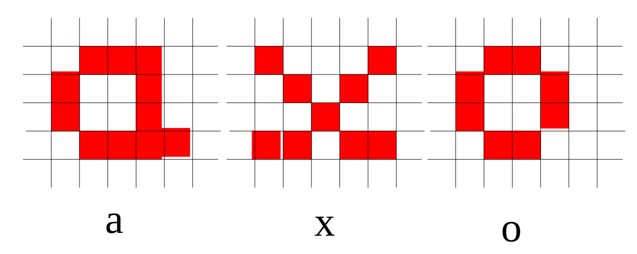
1. ◖ permet à l'utilisateur de dessiner son schéma UML en lui offrant les composants UML utilisés dans le diagramme des classes, et le sauvegarder.
2. ◖ génère automatiquement les différents fichiers java correspondant à votre fichier UML.
3. ○permet la liaison avec des classes standards ou des classes additionnelles dont vous n'avez que le bytecode (regardez le résultat de javap par exemple)
4. ○conversion inverse : depuis une classe ou un source java, générer le diagramme UML
5. ☼intégration d'un éditeur de texte et construction d'un IDE.



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

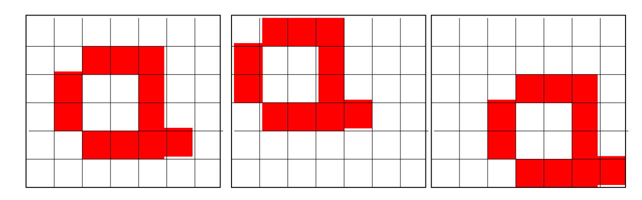
7- Reconnaissance d'écriture**

Le but est d'écrire un programme simplifié de reconnaissance de caractères. Soit une base (un tableau, hashmap, ou autre structure) permettant de stocker et mettre en correspondance pour chaque lettre de l'alphabet, un motif inscrit dans un grille de taille prédéterminée. Par exemple, dans la figure ci-dessous, les lettres a, x et o sont ainsi stockées.

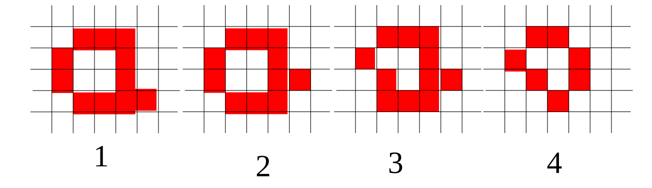


Il s'agit d'offrir ensuite à l'utilisateur une interface (une grille similaire) permettant de dessiner ses propres lettres. Votre travail consistera à trouver la lettre que l'utilisateur aura voulu écrire par comparaison avec les lettres stockées dans la base.

Attention, comme montré ci-dessous, l'utilisateur ne centre pas forcément correctement sa lettre dans la grille, ainsi la même lettre 'a' pourra être reconnue indépendamment de sa position sur la grille (il s'agira d'effectuer les opérations de translation nécessaires, on ne considèrera pas les opérations de rotation et d'échelle).



Evidemment, l'utilisateur n'écrira en général pas exactement la lettre telle que stockée dans la base, c'est ainsi que pour la lettre 'a', les variations telles que représentées ci-dessous pourront être observées.



Ainsi, sur l'exemple, les deux premières lettres peuvent être aisément reconnues comme étant des 'a', l'ambiguité entre la lettre 'o' et la lettre 'a' est par contre possible quand au deux dernières lettres (surtout la dernière lettre).

Pour résoudre cela, il faudra calculer une probabilité de reconnaissance de la lettre (par exemple, dans le cas 1, la lettre est reconnue à 100% en tant que lettre 'a' alors que dans le cas 4, la lettre est reconnue à 49% en tant que 'a', 48% en tant que 'o', et à 3% en tant que lettre 'x'.) La lettre reconnue est ensuite proposée à l'utilisateur qui valide ou non la réponse, et si non, la corrige (en tapant la vraie lettre sur le clavier).

Vous vous efforcerez alors de faire "apprendre" à votre programme la nouvelle manière d'écrire cette lettre, afin que ce même utilisateur re-écrivant cette même lettre la fois d'après a plus de chance de voir sa lettre reconnue. Il ne s'agit pas de remplacer la lettre originale dans la base, mais d'apprendre les variations possibles de la lettre. De la même façon, la base peut éventuellement être vide au départ et alimentée au fur et à mesure qu'elle "apprend" de l'utilisateur.

Dans un second temps, vous étendrez votre programme afin de réaliser une reconnaissance de texte en supposant que l'utilisateur écrit ses caractères de façon bien "déliées".

Pour faciliter la démonstration de l'apprentissage de vos logiciels, les poids (coefficient ou pourcentage) de reconnaissance de chaque lettre seront affichés au fur et à mesure de la reconnaissance.

Vous écrirez un programme qui :

1. ◖ offre à l'utilisateur une interface permettant de dessiner ses propres lettres et qui trouvera la lettre que l'utilisateur aura voulu écrire par comparaison avec les lettres stockées dans la base.
2. ○apprendra de l'utilisateur les variations possibles de la lettre.
3. ○réalise une reconnaissance de texte en supposant que l'utilisateur écrit ses caractères de façon bien "déliées".
4. ○est capable d'intégrer le changement d'échelle et de légère rotation de la lettre lors de la reconnaissance des lettres
5. ☼utilise plusieurs bases actualisées par différents utilisateurs en privilégiant la base de caractères propres à un utilisateur lorsque c'est celui-ci qui écrit.
6. ☼le programme pourra également "deviner" certaines des lettres ambigües en reconnaissant une partie du mot et en cherchant cette partie de mot dans un dictionnaire



T.T. Dang Ngoc
dntt@u-cergy.fr

8- Les souris**

Soit une grille de $N \times N$ cases sur laquelle évolue des souris. Sur cette grille sont également disposés quelques obstacles (que les souris ne peuvent franchir) et des sources de nourritures.

Les souris ont une vision limitée à quelques cases autour d'elles mais ont une excellente mémoire. Elles se rappellent donc de tous les endroits qu'elles ont déjà visitées.

Des sources de nourriture plus ou moins importantes apparaissent aléatoirement et spontanément au cours du temps sur la grille. Chaque source est limitée (pour simplifier, on parlera en nombre d'unités de nourriture). Une souris consomme exactement une unité de nourriture. Une unité de nourriture permet à une souris de survivre pendant t tours de jeu. Au delà de ce temps, si la souris n'a pas mangée, elle meurt.

Il est donc indispensable pour la survie d'une souris qu'elle se dirige vers une source de nourriture afin de manger avant l'expiration de son temps.

Les sources de nourriture n'étant pas inépuisables, il est vital pour les souris d'explorer régulièrement la grille afin de trouver d'autres sources de nourriture, et de veiller au cours de leur exploration d'être toujours à portée d'un point de nourriture connu afin d'y retourner s'il le faut.

Enfin, les souris croisant une autre souris (sur la même case ou une case voisine) peuvent communiquer. Les souris peuvent communiquer leurs connaissances quand à l'emplacement connu de nourriture.

À chaque tour de jeu, chaque souris choisira de se déplacer d'une case ou de rester sur place. Suivant la case où elle se trouvera, elle pourra manger ou communiquer. Il est bien évident que plus une source de nourriture sera utilisée par les souris, plus elle s'épuisera vite.

Chaque souris a un comportement qui lui est propre :

Pour la diffusion des informations, on distinguera :

- les souris coopératives : qui donnent leurs informations aux souris croisées. On pondèrera dans cette catégorie un **degré de fiabilité** passant de honnête (la souris donne toujours les vraies infos) à menteuse (la souris donne systématiquement les infos erronées).
- les souris égoïstes : qui ne fournissent aucune information ;

Pour la réception des informations, on distinguera :

- les souris réceptives : qui tiennent compte des informations qu'on leur communique. On pondèrera dans cette catégorie un **degré de confiance** cela va de naïve (qui croient toutes les informations reçues) à fortement sceptiques (qui croient exactement le contraire de ce qu'on leur dit) ;
- les souris nihilistes : qui ne tiennent pas compte des informations reçues ;

Le but du programme à réaliser est de :

1. ◖ Initialiser une grille :
   - soit de manière aléatoire suivant certains paramètres donnés par l'utilisateur (densités des obstacles, fréquence d'apparition de la nourriture et quantité, nombre de souris)
   - soit de manière manuelle par l'utilisateur.
2. ◖ Exécuter tour par tour la mise à jour de la grille (action des souris, apparition/épuisement des gisements de nourritures)
3. ◖ permettre à l'utilisateur de régler plus finement le comportement des souris (degré de coopération et degré de confiance en fonction de la taille et du nombre de gisement de nourriture, de sa propre faim, du nombre de fois où elle a été induite en erreur, etc.)
4. ○une souris bien nourrie pendant un certain nombre de tours donne naissance à une souris de comportement identique (sans besoin de partenaire !). Permettez la simulation afin de montrer l'évolution de la population au fil des reproductions.
5. ☼pour être plus réaliste, il faut que deux souris (un mâle et une femelle) se rencontrent pour donner naissance à une nouvelle souris (ayant acquis un comportement aléatoirement choisi parmi ceux de ses parents). Le mâle doit avoir été nourris un minimum pour s'accoupler. La femelle doit quand à elle être bien nourrie pendant toute la durée de la gestation (un certain nombre de tour) pour pouvoir donner naissance à n souriceaux.

6. ☼la souris peuvent avoir une certaine mémoire sur les souris qui leur ont déjà menti ou non (et donc tenir compte des informations pour la fois d'après), ont été coopératives avec eux ou non (et leur rendre la pareille), etc.

Mots-clefs : Agent réactif simple coopérants et égoistes



Licence 2 I

2010–2011

Génie logiciel

T.T. Dang Ngoc

dntt@u-cergy.fr

9- Recherche de nourriture par une colonie de fourmis**

Les algorithmes de colonies de fourmis sont des algorithmes inspirés du comportement des fourmis et qui constituent une famille de métaheuristiques d'optimisation. Des biologistes ont ainsi observé, dans une série d'expériences menées à partir de 1989, qu'une colonie de fourmis ayant le choix entre deux chemins d'inégale longueur menant à une source de nourriture avait tendance à utiliser le chemin le plus court.

Un modèle expliquant ce comportement est le suivant :

1. une fourmi (appelée « éclaireuse ») parcourt plus ou moins au hasard l'environnement autour de la colonie ;
2. si celle-ci découvre une source de nourriture, elle rentre plus ou moins directement au nid, en laissant sur son chemin une piste de phéromones ;
3. ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste ;
4. en revenant au nid, ces mêmes fourmis vont renforcer la piste ;
5. si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste ;
6. la piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive ;
7. la longue piste, elle, finira par disparaître, les phéromones étant volatiles ;
8. à terme, l'ensemble des fourmis a donc déterminé et « choisi » la piste la plus courte.

(Source : http ://fr.wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis)

L'objectif de ce projet est de simuler une colonie de fourmis sur un terrain comprenant :
- le nid de fourmi
- des sources de nourriture apparaissant aléatoirement sur le terrain et dont la quantité de nourriture est variable (une "unité de nourriture" désigne ce qui est transporté en une seule fois par une fourmi).
- des obstacles

Les fourmis explorent leur territoire en émettant une "unité de phéromone" tout au long du chemin qu'elle parcourt. Leur parcours est aléatoire mais est influencé par les quantités de phéromones rencontrés. Les phéromones s'évaporent (décrémentent) progressivement au cours du temps (à chaque itération du système).

Le but du programme à réaliser est de :

1. ◖ Initialiser une grille :
   - soit de manière aléatoire suivant certains paramètres donnés par l'utilisateur (densités des obstacles, fréquence d'apparition de la nourriture et quantité, nombre de fourmis)
   - soit de manière manuelle par l'utilisateur.
2. ◖ Exécuter tour par tour la mise à jour de la grille (action des fourmis, apparition/épuisement des gisements de nourritures, piste plus ou moins renforcée des phéromones, nombre d'unités de nourriture ramenées au nid).
3. ◖ la présence d'obstacle de différentes formes pourront être posé par l'utilisateur, et le contournement de l'obstacle par le plus court chemin devront être une conséquence naturelle et statistiques de vos algorithmes.
4. ○Vous améliorerez le programme afin de considérer plusieurs types de nourriture : par exemple : le miel étant plus nourrissant que la viande lui même plus nourrissant qu'un morceau de pomme, une "unité de miel" transportée par une fourmi représentera 3 unités de nourriture, alors que l'"unité de viande" n'en représentera que 2 et la pomme 1. Les fourmis en nombre limité privilégient la goutte de miel, en plus petite quantité mais plus intéressante que la viande et plus encore que la pomme (les unités les plus "riches" étant en général présents en moindre quantité.) Ce problème est connu sous le nom "problème du sac à dos".
5. ☼intégrez une notion de transport coopératif : certains types de nourriture (ne peuvent être découpés sur place et sont tellement lourds qu'il faut plusieurs fourmis pour les transporter)

Mots-clefs : Colonie de fourmi, métaheuristique, problème du sac à dos



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

10- Indicateur d'itinéraire pour GPS dans un réseau de transport**

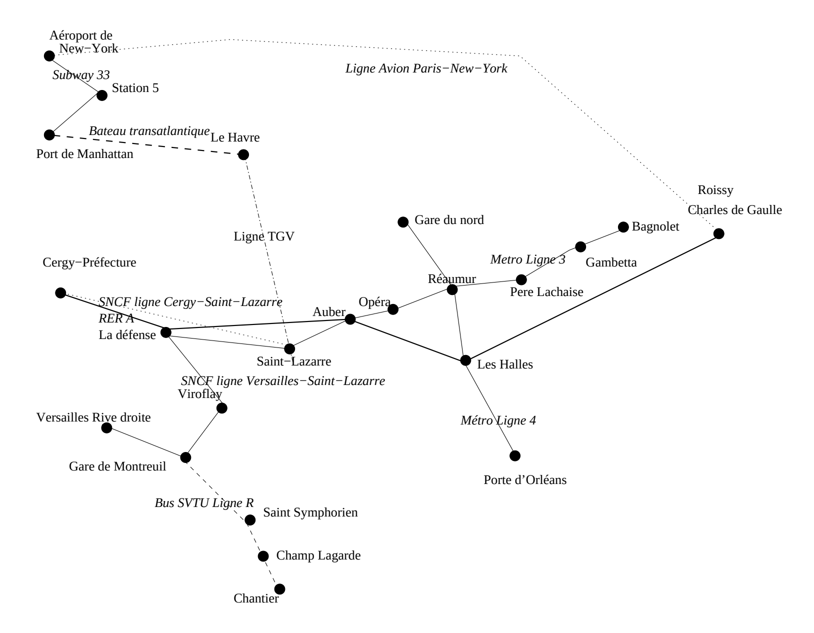
Le but de ce projet est de réaliser un indicateur d'itinéraire pour GPS de poche exploitant le réseau de transport.

Les moyens de transport considérés sont :
- les transports individuels :
  - sans voie : la marche à pied ;
  - avec voie : le vélo ;
  - avec voie : la voiture ;
- les transports en commun :
  - le bus ;
  - le métro ;
  - le train ;
  - le bateau ;
  - l'avion

Un plan de transport est composé de stations (ou appellera station, tout point d'arrêt par lequel transitent les transports en commun et de lignes reliant ces stations.

Une station est décrite par :
- un nom ;
- un type (arrêt d'autobus, station de métro, gare, port naval, aéroport) ;
- des coordonnées le représentant de manière absolu sur une carte du monde (par exemple, des coordonnées GPS) ;

Des lignes de transport en commun relient ces différentes stations. Une station peut être située sur plusieurs lignes, et une ligne peut passer par plusieurs stations (voir figure).



Exemple de réseau de transport

Une ligne de transport en commun est décrite par :
- un nom de ligne ;
- le type de transport associé (bus, métro, train, bateau, avion) ;
- la liste des stations par laquelle cette ligne passe ;

Pour son déplacement, une personne part d'un point appelé **point source** à un **point destination**. Ces points (identifiés par un système de coordonnées GPS simplifié : X,Y) ne sont pas nécessairement situé à l'emplacement d'une station ou sur une voie.

Pour rejoindre une station ou une voie, la personne utilise un transport individuel tels que la marche à pied, le vélo ou la voiture.

Le vélo et la voiture font l'objet d'un réseau routier (route, autoroute et pistes cyclables) et peuvent démarrer et s'arrêter sur ces lignes à n'importe quel endroit sans se soucier de stations comme pour les transports en commun.

La marche à pied permet d'aller («à vol d'oiseau» pour simplifier) n'importe où sans se soucier de stations ou de lignes.

Bien évidemment, chacun de ces moyens de transport a ses limitations qui lui sont propres. On considèrera le coût financier et le temps :

- la marche à pied permet d'aller d'un point à un autre sans restriction de station ou de suivi de ligne ; l'inconvénient étant sa faible vitesse. Son coût financier est nul ;
- le vélo et la voiture n'a pas la contrainte des stations, mais doivent tout de même suivre la route. Le vélo étant bien entendu moins rapide que la voiture (mais son coût est très inférieur) ;
- les transports en commun ont les contraintes des stations et des lignes, et sont plus ou moins rapide et plus ou moins onéreux. Pour certains déplacements, certains sont inévitables (avion ou bateau pour relier Paris-New-York par exemple. Le bateau étant plus lent mais moins cher).

Pour le coûts, vous considèrez un moyenne réaliste du prix au kilomètre (carburant+frais d'entretien+achat pour le vélo et la voiture, billet pour les transports en commun) et pour les temps de transport, vous considérerez une vitesse moyenne au kilomètre réaliste pour chacun des moyens de transports.

Le but de votre programme est de fournir à l'utilisateur :

1. ◖ un moyen de générer le réseau de transport (stations+ligne) et de le paramétrer (coûts, vitesse moyenne, etc.), manuellement et par fichier « cartes ».
2. ◖ une représentation graphique du réseau de transport (cf figure)
3. ◖ un calculateur d'itinéraire de plus court chemin (en temps)
4. ○un calculateur d'itinéraire suivant les paramètres spécifiés par l'utilisateurs (coût minimum, temps minimum, minimum de marche à pied, pas de voiture, pas de bateau, randonnée exclusive donc que de la marche à pied, etc.)
5. ☼permettre à l'utilisateur de définir des étapes ordonnées ou non ordonnées
6. ☼permettre à l'utilisateur de définir en plus de contraintes (pas d'avion, optimiser distance, etc.) des plages horaires valides pour chacune des étapes non-ordonnées

Mots-clefs : Graphe, Algorithme du plus court chemin



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

11- Simulateur de comportement urbain**

Objectif : Faire évoluer un ensemble d'individus sur un tracé de type urbain en respectant a priori des règles mais avec des individus ayant des comportements plus ou moins déviants de ces règles et des objectifs.

- un espace d'évolution (la ville)
- des tracés (route, rue, chemin, trottoir) orientés (voies à sens-unique, double voies), pondérés (vitesse limitées). Le nombre d'individu sur le tracé influe sur la vitesse de circulation (embouteillage). La définition du support de déplacement (route, voie, ligne séparatrice,), des différents tronçons et des carrefours est importante.
- des individus ayant des comportements, des objectifs de déplacement et des rythmes associés.
- des cibles (restaurant, maison, théâtre, cinéma, école, etc.) de capacités plus ou moins limités (à définir).

Dans un intervalle de temps donné, on fait évoluer le trafic des individus et l'on voit l'évolution de celui-ci à chaque pas de temps. Chaque individu, a un ensemble comportements associés : Par exemple : *Les 5 jours de la semaine, M. Dupond part tous les matins à 7h de sa maison, prend la départementale 307 puis l'allée Saint-Fiacre pour déposer ses enfants à l'école. Ensuite, il reprend la départementale puis l'avenue des Etats-Unis pour se rendre à son travail situé au 45 de cette avenue. Il y reste jusqu'à 18h. À 18h, il reprend sa voiture, et va au restaurant s'il n'y a pas trop de monde où il y reste environ 2h. Enfin, il rentre chez lui. Le week-end, M. Dupond, reste chez lui. Toutefois, le samedi, il part faire les courses au marché à 10h00 pendant 1h. Et le dimanche à 17h, il va parfois au cinéma (2h environ).*

Les règles des individus sont paramètrables. Les individus ont des comportements plus ou moins déviants de ces règles et des objectifs. Si une cible est au maximum de sa capacité, l'individu pourra soit décider d'attendre qu'une place se libère, soit renoncer à cette cible.

Le but du programme à réaliser est de :

1. ◖ Initialiser une ville avec un tracé et un ensemble de cibles.
2. ◖ Initialiser un ensemble d'individu et de comportements associés.
   - soit de manière aléatoire suivant certains paramètres tirés au sort dans une liste d'objectifs.
   - soit de manière manuelle par l'utilisateur.
3. ◖ /○Exécuter pas à pas les actions de l'ensemble des individus.
4. ○/☼Avoir des statistiques sur les différents taux d'occupation des cibles au fil du temps.

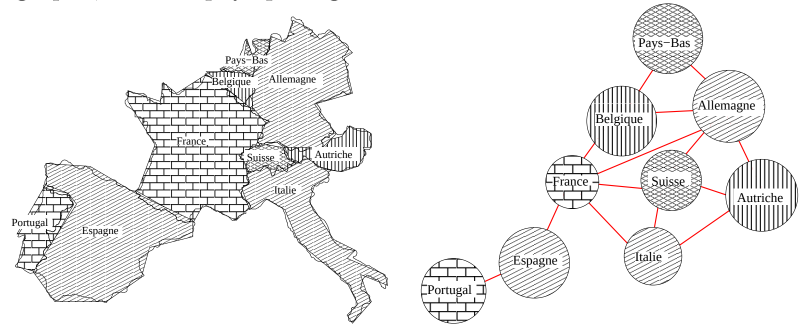
Mots-clefs : Lois de comportement, simulation



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

12- Jeu de conquêtes**

Une carte géographique n'est qu'un ensemble de polygones ayant plus ou moins des frontières communes. Lorsqu'on veut travailler sur les interactions entre pays (coloration, échange, etc.) on représente en général chaque pays comme un sommet d'un graphe, et deux pays partageant une même frontière comme étant relié par un arc.



(a) Carte géographique **(b) Carte sous forme de graphe**

On désire réaliser un jeu stratégie guerrière dont le principe est le suivant :

- Au début de la partie, une carte est générée aléatoirement. Chaque pays partagera certaines frontières avec les pays voisins. Une production par pays est tirée aléatoirement (il s'agit de la production en ressource par tour de chaque pays).
- Chaque puissance se voit attribuer un pays aléatoirement au début de la partie.

À chaque tour, les puissances peuvent réaliser les actions suivantes : créer des soldats, déplacer des soldats, attaquer un pays, réaliser une alliance ou rompre une alliance.

- *Créer des soldats :* : À l'aide des ressources du pays, des armées peuvent être crées (un soldat "coûte" x ressources par tour).
- *Déplacer des soldats :* : Un certain nombre de soldats d'une armée peuvent être déplacé dans un pays adjacent si ce pays appartient à la puissance ou à un de ses alliés.
- *Attaquer un pays :* : Lorsqu'un groupe de soldat est déplacé vers un pays n'appartenant ni à sa puissance, ni à un de ses alliés, il y a alors offensive. Plusieurs puissances peuvent attaquer un même pays en même temps. Le résultat de l'offensive est calculé en comparant le nombre de soldats attaquant le pays multiplié par un coefficient d'attaque et le nombre de soldats se défendant multiplié par un coefficient de défense. À l'issue de la bataille, un certain nombre de pertes est comptabilisés de part et d'autres, et suivant certains critères de victoires, le pays est alors conservé par la puissance défenseur conquis et partagé au pro-rata des attaquants ayant survécu.
- *guerre interne :* : lorsque deux alliances rompent leur pacte, et que leurs armées se trouvent sur le même pays, ils peuvent se faire la guerre. Dans ce cas, l'issu du combat se déroule en calculant la différence du nombre de soldats de chaque puissance sur le pays multiplié par un coefficient multiplicateur aléatoire tiré pour chaque puissance en présence. Le nombre de victimes dans chaque camps est alors décompté pour chaque partis. Les perdants se retirent dans les pays de la même puissance les plus proches. Les gagnants restent sur place et gardent le pays.

Le but de votre programme devra être de :

1. ☾ initialiser une carte : soit manuellement par l'utilisateur, soit aléatoirement suivant des paramètres donnés par l'utilisateur (nombre de pays, nombre de puissances, production de chaque pays, etc.) Une représentation minimale sous forme de graphe est demandée.
2. ☾ permettre de jouer contre d'autres joueurs (sur la même machine, chacun son tour) ;
3. ○permettre de jouer contre l'ordinateur ;
4. ☼vous pourrez améliorer le jeu en définissant des unités différentes sur la carte (char, fantassin, etc. de caractéristiques différentes).

5. ☼vous donnerez une bonne représentation visuelle : représentation des polygones constituant les pays, colorisation des pays, représentation des armées, etc..

Mots-clefs : Graphe



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

13- Création d'un mini SGBD relationnel avec mini-SQL**

Un système de gestion de bases de données (SGBD) permet aux utilisateurs de stocker des données de façon structurée pour pouvoir ensuite les interroger suivant certains critères pour récupérer leurs données. Un SGBD relationnel fait intervenir des tables (ou relations) composées de lignes (tuples) et de colonnes (attributs). Par exemple, les tables *Personnes* et *Villes* suivantes :

Personne			
Prenom	Nom	Age	Adresse
John	Doeuf	21	Paris
Harry	Cover	18	Cergy
Rose	Well	21	Cergy
Jean	Breille	24	Cergy
Jacques	Selère	27	Versailles

et

Ville		
Département	Ville	Code postal
Seine-et-Marne	Provins	77160
Val-d'oise	Cergy	95800
Val-d'oise	Pontoise	95300
Yvelines	Versailles	78000
Yvelines	Conflans	78700
Pas-de-Calais	Calais	62100
Hauts-de-Seine	Meudon	92190

On appelle métadonnées le nom des colonnes du tableau. Par exemple, les métadonnées du tableau personne sont :

Personne			
Prenom	Nom	Age	Adresse

stocker et recharger des données sur le disque

Le SGBD que vous aurez à programmer devra aussi permettre de pouvoir interroger les tables ainsi créées. Pour cela on utilisera une simplification du langage d'interrogation nommé SQL (*Structured Query Language*). Basiquement, ce langage se décrit de la manière suivante :

SELECT *projection*
FROM *tables*
WHERE *condition*

Par exemple :

```
SELECT P.Prenom, P.Age
FROM Personne P
WHERE
      P.age >= 20
  AND P.Adresse = "Cergy"
```

Cet exemple veut dire : "je cherche les noms et les âges des personnes qui ont plus de 20 ans et qui vivent à Cergy. La réponse sera alors :

Table Résultat	
Nom	Age
Well	21
Breille	24

On peut également lier plusieurs tableaux (jointure) par l'utilisation de jointure permettant de lier deux attributs, par exemple :

```
SELECT P.Nom, V.departement
FROM Personne P, Ville V
WHERE
      P.age >= 20
  AND P.Adresse = V.Ville
```

Cet exemple veut dire : "je veux le nom et le département des personnes qui ont plus de 20 ans". La réponse sera alors :

Table Ville	
Nom	Département
Doeuf	Paris
Well	Val-d'oise
Breille	Val-d'oise
Selère	Yvelines

La création et suppression se feront également en utilisant le langage SQL simplfié :

Création d'une nouvelle table :

```
CREATE TABLE "nom de table" ("colonne 1" "type de données pour la colonne 1",
                             "colonne 2" "type de données pour la colonne 2",
```

Ajout d'une donnée dans une table :

```
INSERT INTO "nom de table" ("colonne 1", "colonne 2", ...)
VALUES ("valeur 1", "valeur 2", ...)
```

Suppression de certaines données contenues dans une table :

```
DELETE FROM "nom de table"
WHERE {condition}
```

Modification de certaines données contenues dans une table :

```
UPDATE "nom de table"
SET "colonne 1" = [nouvelle valeur]
WHERE {condition}
```

Modification du schéma d'une table : Ajouter une colonne :

```
ALTER TABLE "nom de table" ADD "colonne 1" "type de données pour la colonne 1"
```

Modification du schéma d'une table : Supprimer une colonne :

```
ALTER TABLE "nom de table"  DROP "colonne 1"
```

Modification du schéma d'une table : Changer un nom de colonne :

```
ALTER TABLE "nom de table"
CHANGE "vieux column name" "nouvelle nom de colonne name"
       "type de données pour le nouveau nom de colonne"
```

Modification du schéma d'une table : Changer le type de données d'une colonne :

```
ALTER TABLE "nom de table"  MODIFY "colonne 1" "nouvelle type de données"
```

Le programme consistera à implémenter le SGBD en :

1. ◖ d'implémenter ce langage SQL simplifié permettant à l'utilisateur de créer les tables de son choix, et de les remplir, de supprimer des tables ou des lignes et d'effectuer des requêtes d'interrogation.
2. ◖ sauvegarder la base sur disque et de la recharger.
3. ○L'accent sur les performances de votre système de gestion de base de données en temps d'exécution et en place mémoire prise sera mis. Aussi, vous considererez (et montrerez lors de la démonstration), des requêtes (dont des jointures) entre des tables de minimum 5.000 lignes.
4. ○gérer les "undo/redo" sur n opérations.
5. ☼d'afficher graphiquement les tables et leurs contenus et d'effectuer les mêmes opérations qu'avec SQL mais de manière graphique.

Mots-clefs : SGBD, bases de données relationelle, algèbre relationnelle, SQL



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

14- Création d'un mini SGBD relationnel avec interface QBE**

Un système de gestion de bases de données (SGBD) permet aux utilisateurs de stocker des données de façon structurée pour pouvoir ensuite les interroger suivant certains critères pour récupérer leurs données. Un SGBD relationnel fait intervenir des tables (ou relations) composées de lignes (tuples) et de colonnes (attributs). Par exemple, les tables *Personnes* et *Villes* suivantes :

Table Personne			
Prenom	Nom	Age	Adresse
John	Doeuf	21	Paris
Harry	Cover	18	Cergy
Rose	Well	21	Cergy
Jean	Breille	24	Cergy
Jacques	Selère	27	Versailles

et

Table Ville		
Département	Ville	Code postal
Seine-et-Marne	Provins	77160
Val-d'oise	Cergy	95800
Val-d'oise	Pontoise	95300
Yvelines	Versailles	78000
Yvelines	Conflans	78700
Paris	Paris	75000
Haut-de-seine	Meudon	92190

On appelle métadonnées le nom des colonnes du tableau. Par exemple, les métadonnées du tableau personne sont :

Table Personne			
Prenom	Nom	Age	Adresse

Le SGBD que vous aurez à programmer devra aussi permettre de pouvoir interroger les tables ainsi créées. Pour cela on utilisera un langage d'interrogation graphique nommé QBE (*Query By Example*). Le langage QBE est très simple, il s'agit de remplir le tableau des métadonnées du tableau qu'on veut interroger avec les contraintes (ou restriction) demandées pour l'attribut. On sélectionnera les colonnes qu'on veut retourner (en couleur sur la figure) dans le résultat final (ou projection).

Par exemple la requête QBE suivante :

Table Personne			
Prenom	Nom	Age	Adresse
		> 20	Cergy

Cet exemple veut dire : "je cherche les noms et les âges des personnes qui ont plus de 20 ans et qui vivent à Cergy. La réponse sera alors :

Table Résultat	
Nom	Age
Well	21
Breille	24

On peut également lier plusieurs tableaux (jointure) par l'utilisation de flêches permettant de lier deux attributs que l'on veut comparer, par exemple :

Table Personne

Prenom	Nom	Age	Adresse
		> 20	●

Table Ville

Département	Ville	Code postal
	●	

Cet exemple veut dire : "je veux le nom et le département des personnes qui ont plus de 20 ans". La réponse sera alors :

Table Resultat	
Nom	Département
Doeuf	Paris
Well	Val-d'oise
Breille	Val-d'oise
Selère	Yvelines

Les flèches pourront être orientés (thêta-jointure) et annoté de sorte à réaliser non seulement des jointures portant sur l'égalité des attributs mais aussi d'autres critères (par exemple "$T1.a > T2.b$ Enfin une interface d'aggrégat (min, max, avg) devra également être proposée (ex. je veux le nombre de personnes qui ont plus de 20 ans et qui habitent Cergy".

Le programme consistera à implémenter le SGBD en :

1. ☾ permettant à l'utilisateur de créer les tables de son choix, et de les remplir mais aussi de supprimer des tables ou des lignes.
2. ☾ définir une interface QBE pour interroger les tables sur des critères de projection, sélection, jointure (equi-jointure et théta-jointure), et d'aggrégats.
3. ☾ sauvegarder la base sur disque et de la recharger.
4. ○L'accent sur les performances de votre système de gestion de base de données en temps d'exécution et en place mémoire prise sera mis. Aussi, vous considererez (et montrerez lors de la démonstration), des requêtes (dont des jointures) entre des tables de minimum 5.000 lignes.
5. ○permettre à l'utilisateur définir un index sur des colonnes, permettant d'accélérer les recherches utilisant des sélections sur ces attributs.
6. ☼permettre à l'utilisateur d'exprimer des contraintes plus précises sur les types de valeurs autorisées. Par exemple, un âge doit être compris entre 0 et 130 ans, un numéro de téléphone doit comporter 5 groupes de 2 chiffres. Lorsque les tables seront remplies, ces contraintes devront être vérifiées.
7. ☼permettre un nombre d'attributs multiples : par exemple, une même personne peut avoir 3 prénoms et 0, 1 ou 2 numéros de téléphone.

Mots-clefs : SGBD, bases de données relationelle, QBE, algèbre relationnelle



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

15- Réalisation d'un mini-moteur de recherche**

Un moteur de recherche est un logiciel permettant de retrouver des ressources (pages Web, forums Usenet, images, vidéo, etc.) associées à des mots quelconques.

Un moteur de recherche est constitué :

- de « robots » (agents, crawler, spiders), qui parcourent les sites à intervalles réguliers et de façon automatique pour découvrir de nouvelles adresses (URL). Ils suivent les liens hypertextes (qui relient les pages les unes aux autres) rencontrés sur chaque page atteinte.
- d'index qui repertorient chaque page visitée suivant des mots-clés
- d'une interface cliente qui permet à l'utilisateur d'interroger l'index suivant un (ou des) mots-clefs afin de retrouver l'URL des pages concernées.

L'index des mots-clefs peut être

- exhaustif : à chaque nouveau mot trouvé dans une page, ce mot est ajouté à l'index
- défini par un dictionnaire : une liste de mots-clefs prédéfinis sera donné au robots, et ces mots constitueront l'index. Cette liste est appelée un dictionnaire.
  Dans ce cadre, il est utile de pouvoir regrouper les mots-clefs en "famille". Par exemple, les mots : programme, programmes, programmeront, programmeur, reprogrammer, reprogrammeur, correspondront à la même entrée dans le dictionnaire.
- catégorisation sémantique : on peut encore aller plus loin et définir des catégorisation par synonyme : manger, dévorer, avaler. Ou par famille sémantique : java, programme, langage, C, C++, bug.

Il peut y avoir plusieurs index permettant ainsi d'exprimer plusieurs types de recherche.

Le but du programme à réaliser est de :

1. ◖ créer un robot qui parcourera récursivement sur N niveaux, une liste d'URL donnée en initialisation du programme. Ce robot indexera les URL suivant les occurences des mots qui seront trouvés dans les pages web correspondantes.
2. ◖ fournir une interface cliente permettant d'interroger l'index suivant un ou plusieurs mots clefs afin de retrouver la(les) pages correspondantes.
3. ○implémenter une procédure permettant de classer les résultats par pertinence (qu'est ce qu'un résultat pertinent ?).
4. ○créer un index défini par un dictionnaire
5. ☼créer un index défini par catégorisation sémantique

Références :

`http://interstices.info/jcms/c_47076/comment-google-classe-les-pages-web`

Mots-clefs : Moteur de recherche, Indexation, Sémantique



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

16- Création d'une table pour jeu de go**

Un plateau de go (ou goban) est composé de N lignes et N colonnes formant ainsi $N \times N$ intersections. Le jeu de go se joue à deux joueurs chacun ayant une couleur de pions nommés pierre (noir ou blanc). Alternativement, chaque joueur place une pierre de sa couleur sur une intersection vide. Un joueur peut passer s'il le veut. Une chaîne de pierres est un ensemble de pierres de même couleur placées de façon contigüe. Lorsque cette chaîne est fermée, elle délimite un espace.

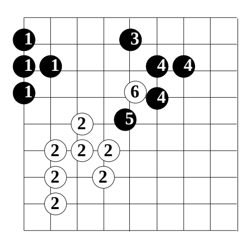


Fig. 3 – on distingue 6 chaînes dans cet exemple

Si aucune pierre adverse ne se trouve dans cet espace, on dit alors que c'est un territoire du joueur qui l'a délimité. Le but du jeu est d'obtenir le maximum de territoires.

Lorsqu'une chaîne de pierres est encerclée par des pierres ennemies sans espace de liberté (c'est-à-dire sans possibilité de continuer sa chaîne), la chaîne de pierre est dite morte et est enlevée du goban.

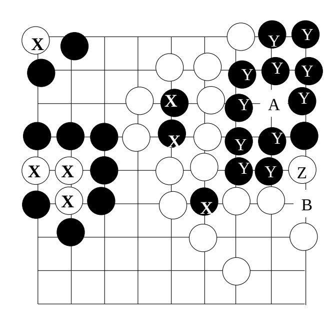


Fig. 4 – Pierres mortes à enlever

Dans la figure, toutes les pierres repérées par 'X' sont mortes et doivent être enlevées aussitôt repérées. On remarquera l'amas noir repéré par des 'Y'. Si c'est à blanc de jouer, en jouant en 'A', il encercle bien noir et fait disparaître toutes les pierres marquées 'Y'. Si c'était à noir de jouer, il aurait joué en 'B' prenant ainsi la pierre blanche marquée Z sécurisant ainsi temporairement son groupe. Il est interdit de jouer dans un territoire ennemi si ce faisant, la pierre posée est tout de suite morte, sauf si ce faisant comme dans notre exemple, cela libère la pierre.

Il existe un cas spécial de figure nommé "ko" représenté sur la figure ci-dessous. Dans cette configuration, si c'est au tour de blanc de jouer, il jouera en 'a' et prendra noir. Puis ça sera au tour de noir de jouer et il pourra jouer en 'b' reprenant ainsi la pierre blanche tout juste mise. Comme cette situation risque de se répéter indéfiniment, la règle suivante est définie : Il est interdit de jouer un coup qui revient à la même situation qu'il y a un coup. Dans notre exemple, après le coup de blanc, noir devra jouer ailleurs avant si c'est encore possible de jouer en 'b'.

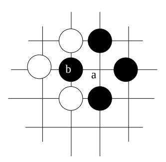


Fig. 5 – Cas du ko

À la fin de la partie (quand un joueur abandonne ou que les joueurs décident d'arrêter d'un commun accord c'est-à-dire qu'ils passent consécutivement tous les deux), le nombre de territoire obtenu est comptabilisé pour chacun des joueurs, c'est à dire le nombre d'intersections vides délimitées par des chaînes. On ajoutera à ces sommes, le nombre de pierres adverses que l'on a pris durant la partie.

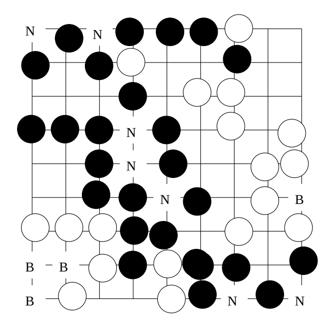


Fig. 6 – Comptage des territoires

Dans cet exemple 9×9, les territoires sont comptés, les 'B' désignent les cases à comptabiliser pour le territoire de blanc et les 'N' ceux du territoire de noir. Blanc a 4 intersections, et noir en a 7. Imaginons qu'au cours de la partie, blanc avait capturé 3 pierres noires et noir 4 pierres blanches. Blanc a $4 + 3 = 7$ et noir a $7 + 4 = 11$. Noir a gagné.

Votre programme devra :

1. ◖ Créer un goban de taille N spécifiée (les valeurs les plus courantes de N étant 19 et 9).
2. ◖ Permettre à deux joueurs de jouer alternativement. Le programme devra être capable de détecter les placements invalides (ko ou suicide de sa propre pierre ou groupe), et de retirer les pierres mortes aussitôt qu'elles sont considérées comme telles..
3. ◖ Compter les points de chaque joueur
4. ○jouer "raisonnablement" contre un utilisateur humain. Note : aucun algorithme vraiment performant permettant à l'ordinateur de gagner contre un utilisateur humain n'existe. Il ne vous sera donc pas demandé d'écrire un algo trop compliqué quant à l'intelligence de l'ordinateur.
5. ☼améliorer "l'intelligence" de votre programme.

Mots-clefs : théorie des jeux



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

17- Déduction

Le but du jeu est de réaliser un jeu permettant de faire deviner à l'ordinateur un animal pensé par l'utilisateur. L'utilisateur pense à un animal, l'ordinateur pose ensuite des questions auxquelles l'utilisateur peut répondre par une des 5 réponses suivantes : oui, probablement oui, ne sais pas, probablement non, et non.
En répondant aux questions, vous permettez à l'ordinateur d'éliminer des ensembles de réponses possibles, mais vous contribuez également à définir votre animal pour les parties suivantes.

Au bout d'une série de questions, l'ordinateur propose sa réponse. L'utilisateur confirme ou infirme la réponse de l'ordinateur. Si la réponse est fausse, l'ordinateur repart pour une autre série de questions. Si au bout de 3 séries, le jeu ne devine pas votre animal, il admet avoir perdu et demande à l'utilisateur la bonne réponse. Il l'entre alors dans sa base de données, se nourrissant ainsi des réponses que vous avez données.

A la fin d'une partie, qu'elle soit gagnante ou perdante pour l'ordinateur, celui-ci demande à l'utilisateur une nouvelle question pertinente et propose ensuite à l'utilisateur d'y répondre pour 10 animaux se trouvant déjà actuellement dans sa base de données.

Exemple de déroulement d'une partie gagnante pour le jeu :

```
Penses à un animal, je vais le deviner.
Ton animal :
- a-t-il des plumes ? non
- a-t-il quatre pattes ? oui
- est il herbivore ? oui
- vit il dans un pays chaud ? oui
- est-il un onglidé ? ne sais pas
C'est un chameau ! non
- vit-il sur le continent africain ? probablement oui
- est-il de couleur uni ? non
- a-t-il des rayures ? non
- a-t-il des taches ? oui
- a-t-il un long cou ? oui
C'est une girafe ! oui
J'ai gagné ! Aide-moi à trouver d'autres questions. Ecris-moi une question :
 vole-t-il ?
Cette question est déjà référencée dans ma base, mais qu'en penses-tu pour ces animaux :
léopard  : non
pingouin : probablement non
chat : non
hibou : oui
zèbre : non
souris : non
truite : non
canard : oui
poule : probablement non
girafe : non
Merci de ta collaboration
```

Exemple de déroulement d'une partie perdante pour le jeu :

```
Penses à un animal, je vais le deviner.
Ton animal :
```

- est il herbivore ? **probablement non**
- a-t-il des plumes ? **non**
- est-il carnivore ? **ne sais pas**
- court-il vite ? **probablement non**
- saute-t-il ? **ne sais pas**

C'est un hérisson ! **non**
- nage-t-il ? **non**
- a-t-il des écailles ? **non**
- a-t-il quatre pattes ? **ne sais pas**
- est-ce un mammifère ? **oui**
- a-t-il des dents ? **oui**
- a-t-il des prédateurs ? **probablement non**
- C'est un blaireau **non**
- a-t-il une bonne vue ? **ne sais pas**
- vit-il dans un terrier ? **non**
- est-il nocturne ? **oui**
- hiberne-t-il ? **ne sais pas**
- vit-il en bande ? **oui**

C'est un loup ! **non**
J'ai perdu ! Quelle était la réponse ?
une chauve-souris
Cet animal n'était pas référencé dans ma base. Je l'enregistre ainsi que tes réponses.
Aide-moi à trouver d'autres questions. Ecris-moi une question :
a-t-il des ailes ?
Cette question n'est pas encore référencée dans ma base, qu'en penses-tu pour ces animaux :
fourmi : **ne sais pas**
antilope : **non**
chauve-souris : **oui**
renard : **non**
pigeon : **oui**
autruche : **oui**
truite : **non**
tortue : **non**
pie : **oui**
saumon : **non**
Merci de ta collaboration

Ecrivez le programme qui :

1. ☾/○permet au jeu de deviner votre animal de s'auto-alimenter des questions et des réponses de l'utilisateur.
2. ☼En utilisant des coefficients statistiques, vous pourrez améliorer votre programme pour permettre une légère tolérance à l'erreur (une erreur de temps en temps de la part de l'utilisateur peut quand même être acceptée).

Références :
http ://fr.akinator.com/

Mots-clefs : Système expert, réseau de neurones, Akinator



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

18- Gestion de réservation de ressources**

Le but est d'écrire un programme de gestion de réservation de ressources.

Il existe de nombreux types de ressources : les ressources humaines (gestionnaires et employés), les ressources informationnelles (information et technologies d'information), les ressources matérielles (équipements, outils, bâtiments), les ressources financières (budget, liquidité, capital-action)...

On désire créer un système permettant de gérer différentes ressources suivant un calendrier avec des ressources exclusives, limitées ou non.

Par exemple dans le cas d'une gestion de cours d'une université, on pourra définir les ressources "salles", une ressource "Enseignant", les promotions d'étudiant et les ressources "Matériel" (vidéo-projecteurs, portables, etc.). Dans cet exemple, les ressources Enseignant, matériel et salles sont exclusives : un enseignant ne peut pas être dans deux salles à la fois, le video-projecteur numéro 3 ne peut pas être utilisé par deux enseignants différents, une salle ne peut pas être utilisée par deux promotions d'étudiants en même temps. Un enseignant peut réserver à la fois un video-projecteur et un portable pour le même créneau, mais pas deux salles.

Dans un autre exemple, celui d'une usine, on pourra prendre comme ressource une énergie (10MW peuvent être utilisés simultanément) , des matériaux (plastique, verre, métal), des machines-outils, du personnel pour gérer la machine. Il faut 3 personnes (A, B et C) pour gérer la machine "fondeuse" qui utilise 3MW/h, 1 tonne de plastique et 2 tonne de verre par heure. 2 autres personnes (D et E) pour la machine "Scierie" qui utilise 5MW/h et 3 tonnes de bois par heure. Au vu des ressources, on peut les faire fonctionner durant le même créneau, mais pas la "découpeuse" qui nécessite 4MW/h, 1 personne (B) et 1 tonne de bois puisque non seulement les MW disponibles seront dépassés mais qu'en plus la personne B est déjà prise par la "fondeuse".

1. ◖ Permettre à l'utilisateur de définir des types de ressources (nom du type, exclusif ou non, limitations, etc.)
2. ◖ Permettre à l'utilisateur de définir des instances de chaque type de ressources
3. ◖ Permettre à l'utilisateur de placer ces ressources sur un calendrier, de suggérer les ressources disponibles au fur et à mesure de la saisie et de signaler les conflits s'il y en a
4. ○De suggérer des créneaux pour placer des combinaisons de ressources prédéfinies
5. ○de placer automatiquement un ensemble de combinaisons de ressources en tenant compte de contraintes (ex. placer les 10 séances de 2h de cours de "Réseaux" des masters fait par l'intervenant XX, sachant qu'il lui faut un vidéo-projecteur et une salle d'au moins 20 personnes, sachant les 8 autres séances de bases de données de ce même master, etc. et sachant que l'intervenant XX fait également des cours de sécurité le jeudi après-midi, etc., autre exemple : sachant qu'il faille produire 3 machines à laver nécessitant l'utilisation de la chaine d'assemblage numéro 42 pendant 2h et 3 employés, sachant la consommation de la chaine 42, sachant les autres éléments en cours de production, etc.)
6. ☼D'optimiser automatiquement le placement des ressources (réalisation en un minimum de temps, en utilisant le moins de budget possible, en effectuant le maximum de taches en parallèle, etc.)

Mots-clefs : Logistique, recherche opérationnelle



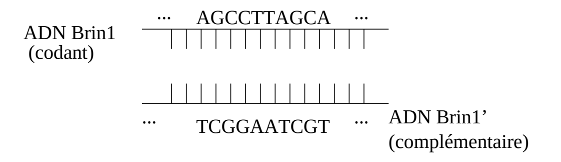
Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

19- ADN**

"L'acide désoxyribonucléique, ou ADN, est une molécule, présente dans toutes les cellules vivantes, qui renferme l'ensemble des informations nécessaires au développement et au fonctionnement d'un organisme. C'est aussi le support de l'hérédité car il est transmis lors de la reproduction, de manière intégrale ou non. Il porte donc l'information génétique et constitue le génome des êtres vivants." (Source Wikipedia)

L'ADN est composé de deux brins se faisant face, et formant une double hélice. L'ADN est composé de 4 nucléotides : la thymine (T), la cytosine (C), l'adénine (A) et la guanine (G) Chaque nucléotide a son nucléotide complémentaire : A-T, T-A, G-C et C-G Un brin d'ADN est composé d'une combinaison des ces 4 nucléotides.

Ainsi, pour un brin d'ADN possédant vingt nucléotides comme dans l'exemple suivant, on peut retrouver la séquence du brin complémentaire et reconstituer la double séquence de la double hélice.



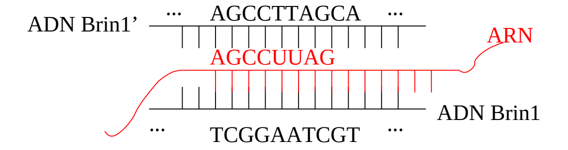
L'information génétique qui constitue le génotype d'un organisme s'exprime pour donner naissance à un phénotype, c'est-à-dire l'ensemble des caractères de cet organisme. Cette expression du génome se fait en interaction avec divers facteurs de l'environnement (nutriments, lumière...). Elle se fait en plusieurs étapes :

1. La transcription, qui est le transfert de l'information génétique de l'ADN vers une autre molécule, l'ARN.
2. La traduction, qui est un transfert d'information depuis l'ARN vers les protéines.

ARN

L'ARN, qui du point de vue de sa structure moléculaire est similaire à l'ADN, se distingue par son rôle essentiel de messager de l'information génétique. L'ARN est un intermédiaire-convoyeur entre l'ADN (dont il copie "en négatif" une séquence d'information) et les structures cellulaires, chargées de lire la séquence d'information copiée de l'ADN en vue de la production des protéines.

(A la différence de l'ADN, l'ARN utilise l'Uracile (U) comme complémentaire de l'Adénine (A). Soit, les combinaisons de complémentarité suivantes : A-U et T-A G-C et C-G



Acide Aminé

Le code génétique est le système de correspondance entre les séquences de nucléotides de l'ADN et les séquences en acides aminés des protéines. Le ribosome est la « machine » assurant la traduction de la molécule d'ARNm dans la synthèse des protéines

Cette traduction est réalisée par triplets de nucléotides : 3 nucléotides codent pour un des 20 acides aminés naturels. Cette correspondance triplet (ou codons) - acide aminé est le code génétique.

Remarque : à un acide aminé peuvent correspondre plusieurs codons (il existe en effet 64 possibilités de codons, et seulement 20 acides aminés).

Le tableau ci-dessous synthétise les correspondances entre codons et acides aminés.

UUU : phénylalanine	UCU : sérine	UAU : tyrosine	UGU : cystéine
UUC : phénylalanine	UCC : sérine	UAC : tyrosine	UGC : cystéine
UUA : leucine	UCA : sérine	UAA : stop	UGA : stop/sélénocystéine
UUG : leucine	UCG : sérine	UAG : stop	UGG : tryptophane
CUU : leucine	CCU : proline	CAU : histidine	CGU : arginine
CUC : leucine	CCC : proline	CAC : histidine	CGC : arginine
CUA : leucine	CCA : proline	CAA : glutamine	CGA : arginine
CUG : leucine	CCG : proline	CAG : glutamine	CGG : arginine
AUU : isoleucine	ACU : thréonine	AAU : asparagine	AGU : sérine
AUC : isoleucine	ACC : thréonine	AAC : asparagine	AGC : sérine
AUA : isoleucine	ACA : thréonine	AAA : lysine	AGA : arginine
AUG : méthionine/start	ACG : thréonine	AAG : lysine	AGG : arginine
GUU : valine	GCU : alanine	GAU : acide aspartique	GGU : glycine
GUC : valine	GCC : alanine	GAC : acide aspartique	GGC : glycine
GUA : valine	GCA : alanine	GAA : acide glutamique	GGA : glycine
GUG : valine	GCG : alanine	GAG : acide glutamique	GGG : glycine

De plus, 3 triplets ne codent pour aucun acide aminé. Ces triplets "non-sens" indiquent, lors de la traduction, la fin de la protéine. Ils sont ainsi nommés "codons STOP".

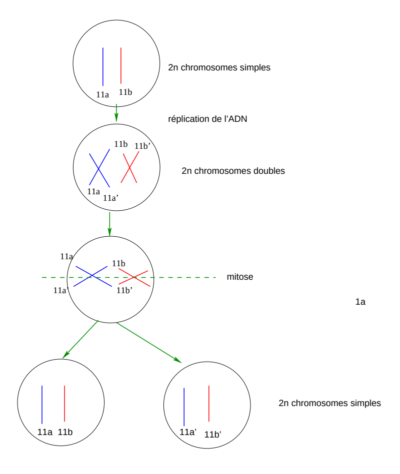
Chromosomes

Le chromosome est l'élément porteur de l'information génétique. Les chromosomes contiennent les gènes et permettent leur distribution égale dans les deux cellules filles lors de la division cellulaire. Ils sont formés d'une longue molécule d'ADN. Entre deux divisions, la séparation entre les différentes molécules d'ADN (chromosomes) est peu perceptible, l'ensemble porte alors le nom de chromatine. Ils se condensent progressivement au cours de la division cellulaire pour prendre une apparence caractéristique en forme de X à deux bras courts et deux bras longs, reliés par un centromère.

Au cours du cycle cellulaire, la cellule est amenée à se diviser soit par mitose soit par meiose.

La mitose est un phénomène général de la division cellulaire. C'est une division unique, asexuée. Son rôle est le renouvellement des cellules mortes, la croissance et la cicatrisation. Elle s'effectue de la manière suivante :

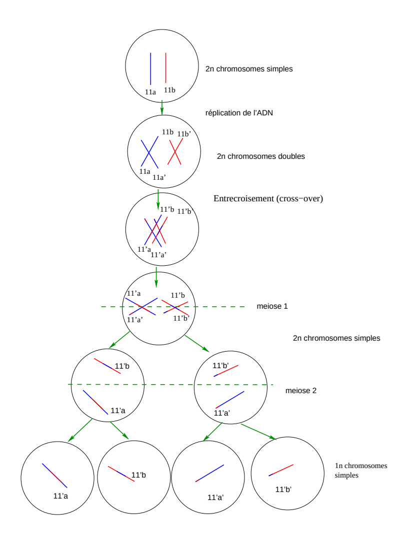
A partir d'une cellule mère comportant 2n chromosomes simples (sur la figure, nous n'avons représenté que la paire de chromosome 11 : 11a et 11b), une réplication de l'ADN a lieu. Chaque chromosome est donc dupliqué (sur la figure, chaque chromosome 11 est dupliqué : 11a est dupliqué en 11a' et 11b en 11b'). Lors de la mitose, la cellule se divise en emportant un réplicat de chaque chromosome. Ainsi, on obtient 2 cellules filles de 2n chromosomes simples chacune (sur la figure, on obtient, la cellule fille contenant la paire de chromosomes 11a et 11b et la deuxième cellule fille contenant la paire de chromosome 11a' et 11b').



Notez que pour des raisons de lisibilité, nous n'avons montré que le cas du choromosome 11 sur la figure, et qu'en réalité, il faudrait y faire figurer toutes les autres paires de chromosomes qui effectuent leur mitose en même temps.

La meiose est un processus aboutissant à la création de cellules sexuelles (gamètes par 2 divisions cellulaires successives. Le rôle est la reproduction. La diversité génétique étant assuré d'une part par l'entrecroisement (cross-over) et la création de 4 cellules filles issues de l'un ou de l'autre chromosome de la paire initiale. Sur la figure, nous n'avons représenté que la paire de chromosome 11 : 11a et 11b. une réplication de l'ADN a lieu. Chaque chromosome est donc dupliqué (sur la figure, chaque

chromosome 11 est dupliqué : 11a est dupliqué en 11a' et 11b en 11b'). Un entrecroisement a ensuite lieu, et des gènes sont passent ainsi d'un chromosome à l'autre (au même emplacement). Formant ainsi les chromosomes 11'a, 11'a', 11'b et 11'b'. Lors de la meiose 1, deux cellules de 2n chromosome simples sont issues (l'une contenant 11'a et 11'b, l'autre contenant 11'a' et 11'b'). Enfin la meiose 2 sépare chaque chromosome et forme ainsi 4 cellules de 1n chromosomes simples (11'a, 11'b, 11'a' et 11'b').



Travail demandé

1. ◖ permette à l'utilisateur de saisir la chaine d'un brin d'ADN
2. ◖ génère le brin complémentaire
3. ◖ réalise la transcription via l'ARN messager puis crée la chaine d'acide aminé résultant
4. ○réalise la duplication de l'ADN par l'intermédiaire de l'ARN
5. ○intégrer l'utilisation de l'ADN dans les chromosomes afin d'effectuer la meiose, la mitose et la fusion. L'utilisateur pourra ainsi "zoomer" sur un chromosome pour étudier la portion d'ADN correspondante en cours de duplication, entrecroisement, etc. Les mutations, durant la duplication pourront être possible (et paramétrables)
6. ☼renseignez-vous sur les introns et les exons afin de pouvoir modéliser (de manière simplifiée) des gènes.

Références :

http://www.adn.wikibis.com/acide_desoxyribonucleique.php
http://www.mon-genome.com/code_genetique.php



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

20- Arbre génétique**

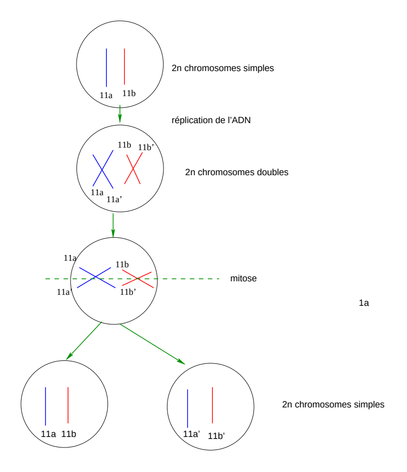
Chromosomes

Le chromosome est l'élément porteur de l'information génétique. Les chromosomes contiennent les gènes et permettent leur distribution égale dans les deux cellules filles lors de la division cellulaire. Ils sont formés d'une longue molécule d'ADN. Entre deux divisions, la séparation entre les différentes molécules d'ADN (chromosomes) est peu perceptible, l'ensemble porte alors le nom de chromatine. Ils se condensent progressivement au cours de la division cellulaire pour prendre une apparence caractéristique en forme de X à deux bras courts et deux bras longs, reliés par un centromère.

Au cours du cycle cellulaire, la cellule est amenée à se diviser soit par mitose soit par meiose.

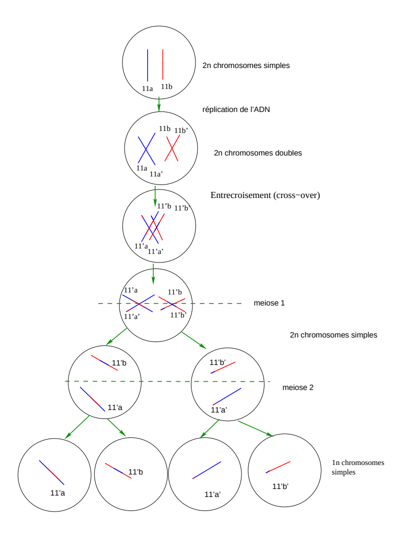
La mitose est un phénomène général de la division cellulaire. C'est une division unique, asexuée. Son rôle est le renouvellement des cellules mortes, la croissance et la cicatrisation. Elle s'effectue de la manière suivante :

A partir d'une cellule mère comportant 2n chromosomes simples (sur la figure, nous n'avons représenté que la paire de chromosome 11 : 11a et 11b), une réplication de l'ADN a lieu. Chaque chromosome est donc dupliqué (sur la figure, chaque chromosome 11 est dupliqué : 11a est dupliqué en 11a' et 11b en 11b'). Lors de la mitose, la cellule se divise en emportant un réplicat de chaque chromosome. Ainsi, on obtient 2 cellules filles de 2n chromosomes simples chacune (sur la figure, on obtient, la cellule fille contenant la paire de chromosomes 11a et 11b et la deuxième cellule fille contenant la paire de chromosome 11a' et 11b').



Notez que pour des raisons de lisibilité, nous n'avons montré que le cas du choromosome 11 sur la figure, et qu'en réalité, il faudrait y faire figurer toutes les autres paires de chromosomes qui effectuent leur mitose en même temps.

La meiose est un processus aboutissant à la création de cellules sexuelles (gamètes par 2 divisions cellulaires successives. Le rôle est la reproduction. La diversité génétique étant assuré d'une part par l'entrecroisement (cross-over) et la création de 4 cellules filles issues de l'un ou de l'autre chromosome de la paire initiale. Sur la figure, nous n'avons représenté que la paire de chromosome 11 : 11a et 11b. une réplication de l'ADN a lieu. Chaque chromosome est donc dupliqué (sur la figure, chaque chromosome 11 est dupliqué : 11a est dupliqué en 11a' et 11b en 11b'). Un entrecroisement a ensuite lieu, et des gènes sont passent ainsi d'un chromosome à l'autre (au même emplacement). Formant ainsi les chromosomes 11'a, 11'a', 11'b et 11'b'. Lors de la meiose 1, deux cellules de 2n chromosome simples sont issues (l'une contenant 11'a et 11'b, l'autre contenant 11'a' et 11'b'). Enfin la meiose 2 sépare chaque chromosome et forme ainsi 4 cellules de 1n chromosomes simples (11'a, 11'b, 11'a' et 11'b').



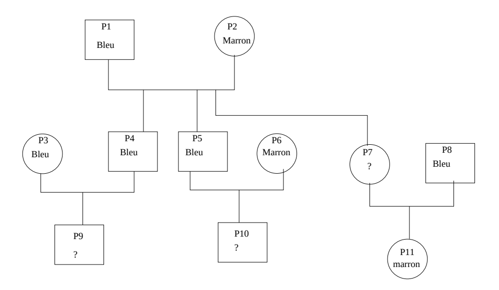
Chaque cellule humaine, excepté les gamètes, possède 22 paires de chromosomes appelés autosomes, numérotées de 1 à 22 par ordre de taille décroissante, et une paire de chromosomes sexuels appelés gonosomes : XX chez la femme et XY chez l'homme. Lors d'une fécondation, les 22 chromosomes + (X ou Y) de l'homme fusionnent avec les 22 de chromosomes + X de la femme. Il en résulte ainsi 22 paires de chromosomes + (X ou Y) dans la cellule qui formera le futur bébé. Notez ainsi que chaque paire de chromosome de l'enfant comportera un chromosome du père et un chromosome de la mère. Le 23ème chromosome transmis par le père (un X ou un Y), déterminera le sexe de l'enfant (XX pour une fille, XY pour un garçon).

Chaque chromosome porte un grand nombre de gènes (codant chacun ou presque, une caractéristique morphologiques, physiologiques, comportementaux). Dûes aux paires de chromosomes, l'information génétique est en double (sauf pour certaines parties des chromosomes sexuels). Chaque copie d'un gène est appelée allèle.

Arbre génétique

D'une manière générale, l'information génétique exprimée résulte de l'expression conjointe des allèles en présence. Un allèle dominant s'exprime toujours dans le génome de son porteur. Cependant, si l'information d'un allèle n'est pas exprimée lorsqu'un allèle dominant du même gène est présent, c'est un allèle récessif. La particularité de l'allèle récessif d'un gène est qu'il peut être présent dans le génome et transmis sur plusieurs générations sans qu'il ne s'exprime dans le phénotype de ses porteurs. S'il n'y a pas d'allèle dominant, les deux exemplaires du gène ont le même allèle récessif (homozygote récessif), alors le caractère récessif est exprimé.

Par l'utilisation d'arbre généalogique, il est ainsi possible de déterminer l'expression d'un gène au sein d'une famille.



Par exemple, si l'on sait que le gène "yeux marrons" est dominant et "yeux bleus" récessif. L'arbre généalogique ci-dessous montre que : Il faut 2 allèles "Yeux bleus" pour avoir les yeux bleus, donc P1, P3, P4, P5 et P8 ont les 2 allèles "Yeux Bleus". Une personne ayant les yeux marrons peut avoir soit les 2 gènes "Marrons" soit 1 gène "Marron" et un gène "Bleu". P3 et P4 ayant tous les allèles bleus, leur fils héritant d'un allèle de P3 et d'un allèle de P4 aura forcément les yeux bleus. P10 aura un allèle bleu de P5 et un allèle de P6 (bleu ou marron si P6 a 1 allèle marron et un allèle bleu, marron si P6 a ses 2 allèles marrons). Soit entre 1/4 et 1/2 chances d'avoir les yeux bleus. P7 a 1 allèle bleu (issu de P1). P8 a les deux allèles bleus. P11 a au moins un allèle marron (puisqu'elle a les yeux marrons) Or elle a un allèle bleu issu de P8, donc l'allèle marron provient de P7. Donc P7 a un allèle bleu et un allèle marron.

Travail demandé

1. ☾ permet de générer une version simplifiée des 23 chromosomes et permettre à l'utilisateur de placer des gènes sur les chromosomes (pour simplifier, on donnera simplement des identifiants aux emplacements des gènes sur les chromosomes) puis de simuler la mitose, meiose et la fusion et visualiser les emplacements des gènes sur les cellules résultantes.
2. ☾ permettre de dessiner des arbres généalogiques génétiques et de déduire des probabilités (ou une certitude) sur l'expression des gènes sur une personne de l'arbre généalogique.
3. ○Certains gènes sont portés par le chromosome sexuel (le 23ème) et donc dans le cas d'un garçon n'est codé qu'en un seul exemplaire. De fait, il devient automatiquement dominant (puisque unique). S'il est sur le X et qu'il est récessif, la mère est dite porteuse et le transmettra (avec une probabilité de 1/2) à son fils qui l'exprimera, ses filles quand à elles, pourront le porter (avec une probabilité de 1/2) sans l'exprimer (puisqu'il est récessif). Les gènes portés par Y sont uniquement transmis de père à fils (avec une probabilité de cent pour cent). Considérez ce cas dans l'arbre généalogique
4. ☼les informations sont incomplètes, on peut se baser sur des fréquences d'une maladie génétique, d'une caractéristique, pour déduire la probabilité d'expression du gène chez un individu. Complétez votre programme. Détectez les anomalies de type : deux personnes aux yeux bleus ont un enfant aux yeux marrons.
5. ○il existe des aberrations chromosomiques (trois chromosomes au lieu d'un, ou au contraire un seul chromosome) dû à une anomalie lors de la meiose. Tenez-en compte lors de votre développement.
6. ☼complétez votre programme pour réaliser des tests de paternité ou de maternité. C'est à dire à partir de génotype de chacun (ou d'un seul) des supposés parents et de l'enfant, votre programme devra déduire avec une certaine probabilité qui est le géniteur ou la génitrice supposée.

Références :

http://fr.wikipedia.org/wiki/Chromosome
www.unites.uqam.ca/pcpes/ppt/e07/mitose.ppt
http://fr.wikiversity.org/wiki/Notions_de_base_en_génétique



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

21- Chaine alimentaire**

Dans un écosystème, les liens qui unissent les espèces sont le plus souvent d'ordre alimentaire. On distingue trois catégories d'organismes :

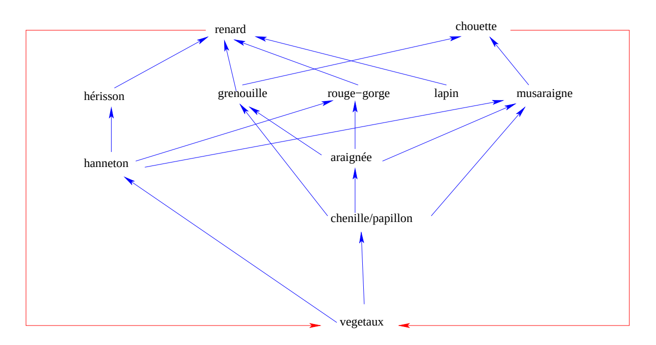
1. les producteurs (surtout les végétaux chlorophylliens, capables, grâce à la photosynthèse, de fabriquer de la matière organique à partir de dioxyde de carbone et de lumière solaire, mais aussi d'autres organismes autotrophes, certains étant à la base de chaînes alimentaires totalement indépendantes de l'énergie solaire) ;
2. les consommateurs (les animaux) ; il existe trois types de consommateurs :
   - les herbivores qui se nourrissent des producteurs, on les appelle aussi consommateurs primaires
   - les carnivores primaires, ou encore consommateurs secondaires, qui se nourrissent des herbivores
   - les carnivores secondaires, appelés également consommateurs tertiaires, qui se nourrissent des carnivores primaires ;
3. les décomposeurs (les bactéries, champignons) qui dégradent les matières organiques de toutes les catégories et restituent au milieu les éléments minéraux.

Ces relations forment des séquences où chaque individu mange le précédent et est mangé par celui qui le suit ; on parle de chaîne alimentaire. Chaque maillon est un niveau trophique. La niche écologique est ce que partagent deux espèces animales quand elles habitent le même milieu et qu'elles ont le même régime alimentaire. Ainsi, deux espèces ayant la même niche sont en «compétition».

Par exemple, dans un écosystème très simple, composé de deux populations, de lièvres et de lynx, jusqu'ici considérées comme isolées l'une de l'autre. Dans ces conditions, la population des lièvres croît exponentiellement et celle des lynx décroît exponentiellement. Mais les lynx sont des prédateurs des lièvres... C'est en capturant des lièvres et en s'en nourrissant qu'ils peuvent se développer. À l'inverse, la population des lièvres est directement affectée par ces captures. L'évolution de l'effectif des lynx et celle des lièvres sont ainsi liées. Plus il y a de proies, plus il est facile pour un prédateur d'en capturer une ; symétriquement, plus il y a de prédateurs, plus les proies sont susceptibles de les rencontrer avec une issue tragique pour elles.

Reconsidérons à présent la croissance exponentielle. À l'évidence, il n'est pas réaliste d'imaginer qu'une population animale puisse croître exponentiellement sans rencontrer à un moment ou à un autre des limites à sa croissance. En effet, elle exploite des ressources qui sont évidemment limitées ; ainsi en va-t-il de l'herbe pour nos lièvres, ou plus simplement encore de la superficie du territoire disponible.

On peut représenter une prédation (par exemple les lynx mangent les lièvres) par un arc orienté. Et donc constituer un graphe orienté représentant la chaine alimentaire. Il faudra évidemment paramétrer les différentes populations (vitesse de croissance, densité maximum) Voici par exemple un ecosystème que l'on pourra définir :



Vous écrirez un programme qui

1. permettra à l'utilisateur de représenter un ecosystème et de le paramétrer (vitesse de croissance, densité maximum)
2. simuler le comportement de l'écosystème en faisant état à chaque tour, du nombre d'individus de chaque population. Vous y considèrerez les producteurs et les consommateurs

3. ○vous ajouterez le comportement des décomposeurs qui se nourissent soit de la décomposition d'animaux morts (donc, il faut que l'animal soit effectivement mort pour les nourrir), soit des secrétions et déjections des animaux (acariens, bousiers, etc.) (dans ce cas là, ils peuvent se nourrir tant que l'animal est vivant...).
4. ○vous enrichirez le paramétrage de vos populations (nombre de calories apportées au prédateur, nombre de calories à ingurgiter chaque jour, ou tout autres paramètres que vous jugerez pertinents pour la simulation)
5. ☼Vous pourrez représenter des territoires où plusieurs ecosystèmes peuvent se "croiser" à certains endroits, où certaines populations doivent strictement se cantonner (les girafes par exemple n'iront pas dans un territoire trop "froid"), et observer la migration de certaines populations (par exemple, on peut imaginer une population de lapins en pays chaud, qui, parce qu'il supportent le froid, migreront petit à petit vers un territoire plus froid pour échapper aux lions qui eux doivent rester au chaud !
6. ☼Vous tenterez de prédire l'issue sur des écosystèmes simples à l'aide de système d'équations différentielles (cf. références interstice)

Références :

http://fr.wikipedia.org/wiki/Réseau_trophique
http://interstices.info/jcms/n_49941/systemes-dynamiques-et-equations-differentielles
http://interstices.info/jcms/i_56750/modeliser-la-dynamique-des-populations-animales-la-predation

Mots-clefs : Simulation, Automates cellulaires

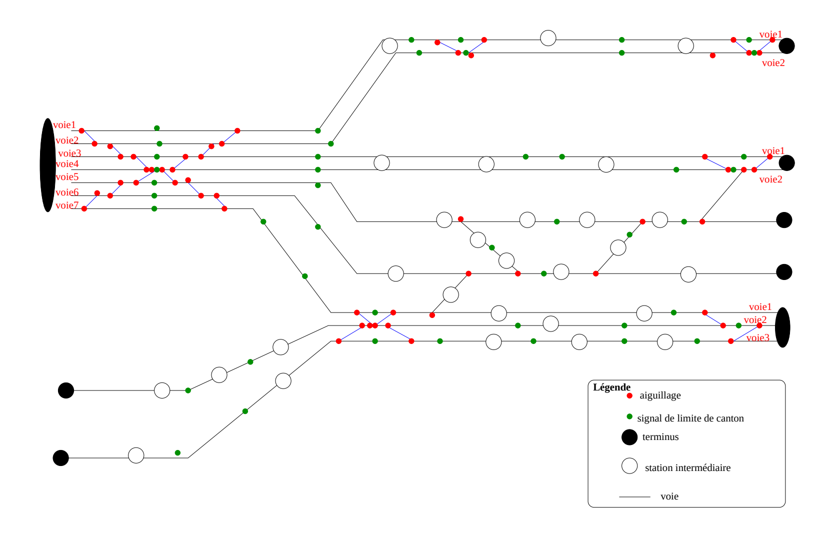


Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

22- Trafic ferroviaire**

On considère des lignes de trains. Chaque ligne de trains a deux extrêmités et plusieurs stations intermédiaires. Une fois arrivé à une extrêmité, un train doit ensuite repartir dans l'autre sens.

Un train circule sur une voie (rails). Les voies peuvent se croiser (aiguillage) et un train peut ainsi passer d'une voie à l'autre. Plusieurs lignes peuvent ainsi avoir des portions de voies communes.



Voies, aiguillage et signalisation

La signalisation ferroviaire est un ensemble de signaux, de dispositifs et de règlements destinés à assurer la sécurité des circulations ferroviaires. Nous nous intéresserons qu'aux risques inhérents à la circulation ferroviaire :

– le « rattrapage », quand le train suiveur rattrape celui qui le précède,
– le « nez à nez », quand deux trains se retrouvent face à face sur la même voie,
– la « prise en écharpe », quand un train arrive sur un aiguillage déjà occupé par un train venant d'une autre direction.

Le risque de rattrapage est pris en charge par le cantonnement ; Le cantonnement est le moyen généralement employé pour assurer l'espacement des trains circulant dans le même sens sur une même voie. Par principe, on n'admet que la présence d'un seul train dans un canton donné. Lorsqu'un train pénètre dans un canton, le signal d'entrée du canton est fermé. Lorsque le train poursuivant sa marche entre dans le canton suivant, le signal d'entrée de ce dernier est fermé tandis que celui du canton précédent est ouvert.

Le risque de nez-à-nez est pris en charge par les enclenchements de sens ; En fonction de la vitesse des trains, on imposera un nombre minimum de cantons intermédiaires (et comportant au moins un aiguillage...) entre deux trains circulant dans des directions opposées sur la même voie.

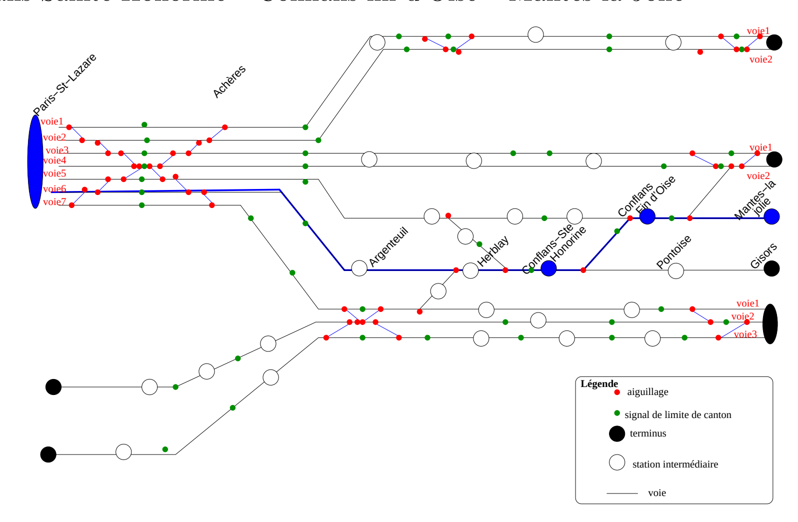
Le risque de prise en écharpe est pris en charge par les enclenchements internes au poste d'aiguillage (enclenchement d'itinéraires, enclenchement de transit...) ;

Lignes

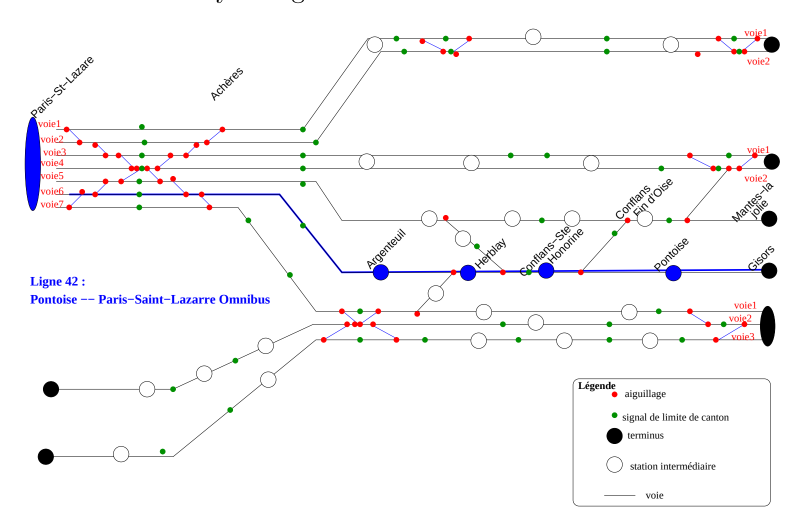
Chaque ligne de trains à deux extrêmités (qui ne sont pas obligatoirement les extrêmités physiques des voies) et plusieurs stations intermédiaires. Les voies peuvent se croiser (aiguillage) et un train peut ainsi passer d'une voie à l'autre. Plusieurs lignes peuvent ainsi avoir des portions de voies communes. Une ligne de train ne passe pas forcément par toutes les stations du parcours (omnibus, direct, semi-direct, etc.). Il peut y avoir plusieurs voies par station.

Dans l'exemple des figures ci-dessous, deux lignes sont représentées,

La ligne 17 : "Paris Saint-Lazare – Mantes-la-Jolie semi-direct" s'arrête aux arrêts suivants : Paris Saint-Lazare – Conflans-Sainte-Honorine – Conflans-fin-d'Oise – Mantes-la-Jolie



Et la ligne 42 : "Pontoise – Paris-Saint-Lazare Omnibus" s'arrête aux arrêts suivants : Pontoise – Conflans-Sainte-Honorine – Herblay – Argenteuil et Paris-Saint-Lazare.



1. ◖ vous permettrez à l'utilisateur de dessiner son plan de voies de trains, les stations intermédiaires et les aiguillages
2. ◖ vous permettrez à l'utilisateur de définir ses lignes : départ, arrivée, stations intermédiaires, et permettrez -sans contraintes horaires- la simulation de plusieurs trains sur chacune des lignes. Votre programme devra gérer les aiguillages et les cantons de sorte à ce que le trafic puisse se faire sans accrochage ni collision.
3. ○vous indiquerez la distance entre chaque station, et partant d'une vitesse définie pour les trains, votre programme devra être capable d'indiquer les heures d'arrivée à chaque station (ie. afficher l'indicateur horaire) sachant les horaires de départ
4. ○vous pourrez évidemment avoir plusieurs trains desservant la même ligne.
5. ○Vous pouvez prévoir des dépôts ou des "voies de garage" permettant de stocker les trains lorsqu'ils ne sont pas utilisés ou pour permettre de dégager une voie le temps qu'un autre train passe. Soyez réaliste en gérant un nombre limité de trains par ligne, et n'oubliez pas que sur une ligne, il n'y a de "retour" que s'il y a eu des "allers" (vous ne pouvez pas envoyer un nombre infini de trains dans le même sens sur une même ligne, il faut bien ramener les trains à un moment...).
6. ☼Faites de l'optimisation. Prévoyez un taux d'affluence moyen par station et par créneau horaire, et trouver comment organiser vos lignes de manière efficace (les aiguillages, cantons, nombre de rames, horaires, stations désservies, etc.)

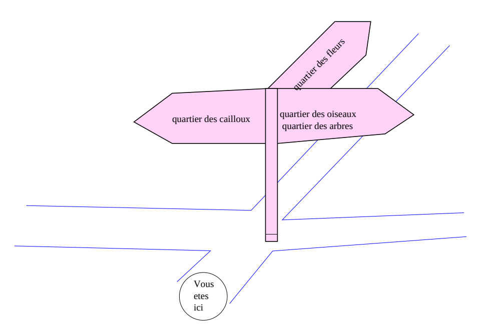
Mots-clefs : Recherche opérationnelle, programmation par contrainte



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

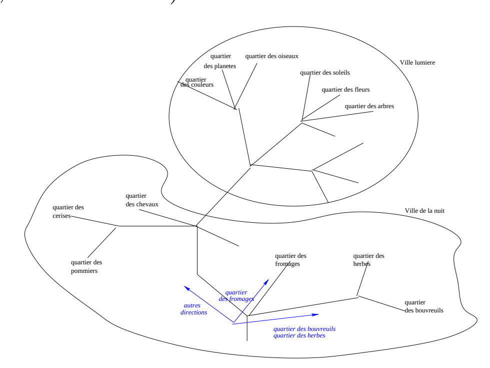
23- Pose de panneaux indicateurs**

Afin d'orienter correctement les personnes cherchant leur chemin (vers une maison par exemple) et n'ayant pas de cartes, on dispose aux intersections des rues, des panneaux indicateurs donnant la direction des quartiers limitrophes. On peut vers une même direction, indiquer un nombre raisonnable de quartiers.



On suppose qu'une fois que la personne est dans le quartier recherché, elle peut sans l'aide de panneaux supplémentaires, trouver la maison qu'elle recherche par son numéro. Au sein d'une ville, on ne peut pas à chaque intersection, indiquer la direction de **tous** les quartiers. Aussi dispose-t-on d'un panneau de route par défaut appelé "autres directions". Il ne peut y avoir évidemment au maximum qu'un seul panneau "autres directions" à chaque intersection.

Au vu des structures des villes, il n'y a en général que quelques grands axes (un seul pour les petits villages) permettant de sortir de la ville (les départementales, les nationales).



De la même manière, quelques grandes nationales permettent de relier les régions, et on considèrera quelques axes internationaux pour relier les pays.

Il est indispensable que les panneaux soient placés de sorte à ce que tout voyageur où qu'il soit, puisse arriver à destination en consultant uniquement les panneaux. Pour des raisons d'économie, il est également indispensable d'économiser au maximum le nombre de panneaux posés à chaque intersection (en utilisant au maximum les panneaux "autres directions").

Les adresses sont hiérarchisées : numéro de rue, nom du quartier, ville, région, pays.

Un voyageur dispose d'une adresse complète pour arriver à destination. Un panneau de direction peut indiquer soit un ensemble de quartiers, un ensemble de villes, un ensemble de région ou un ensemble de pays. Ainsi, sur les intersections des grands axes frontaliers, des panneaux indiquant simplement quels sont les villes (resp. régions, resp. pays) atteint(e)s, permettant ainsi d'aggréger plusieurs quartiers (resp. villes, resp. régions). Par exemple plutôt que de dire que dans la direction de ce grand axe, on atteint *ici, la liste de tous les quartiers de la ville de Paris*, on mettra simplement un panneau indiquant "Ville de Paris".

Il arrive qu'on crée de nouveaux quartiers, les renomme, les supprime (de même qu'on peut créer/renommer/supprimer de nouvelles villes, régions et pays!). Pour certaines raisons (axes fermés), on ne peut plus emprunter une direction pour atteindre ce quartier (villes/région ou pays). De ce fait, cela influe sur un certain nombre de panneaux indicateurs.

Lorsqu'il y a très peu de changement, on peut se permettre d'aller repeindre manuellement quelques panneaux.

Lorsque cela arrive fréquemment et sur beaucoup de localité simultanées, certaines villes choisissent de s'équiper de panneaux indicateurs plus modernes (panneaux automatisés Version 2), qui savent uniquement discuter avec les panneaux voisins (ceux situés à la prochaine intersection). Il est possible de ce fait que les panneaux puisse de proche en proche s'échanger suffisamment d'information pour qu'au bout d'un moment les panneaux de la ville soient tous corrects par rapport à la nouvelle configuration. Et, fait intéressant, en rajoutant simplement une valeur correspondant au nombre de panneaux intermédiaires jusqu'à la direction cherchée, il est possible aux panneaux de déduire un plus court chemin (en terme du "moins de quartiers à traverser") jusqu'à la destination. Si l'automatisation fonctionne bien, la direction de la prochaine intersection à atteindre suivant la meilleure route (le plus court chemin) pourrait être indiquée.

D'autres villes quand à elles, choisissent de s'équiper de panneaux encore plus automatisé et plus intelligents Version 3 (et plus chers), pouvant traiter suffisamment d'information pour être capable de retenir le plan de la ville en partie ou en totalité. Ces plans ne leur étant pas fournis, ils doivent être automatiquement créés et mis à jour par discussion avec les autres panneaux. Du fait de la bonne connaissance du plan des alentours, il devrait être simple d'afficher le plus court chemin vers les destinations fléchées.

1. ◖ vous permettrez à l'utilisateur de créer ses axes routiers et ses différents chemins. Votre programme devra ensuite poser ses panneaux de façon optimale (ne pas multiplier le nombre de directions indiquées inutilement en utilisant au maximum les panneaux "autres directions") et permettre à n'importe quel voyageur d'arriver à destination quelque soit sa provenance et quelque soit sa destination. Vous implémenterez dans un premier temps, uniquement les panneaux de bases (Version 1).
2. ◖ vous permettrez à l'utilisateur de placer un voyageur dans n'importe quel quartier du monde, et muni simplement d'une adresse de destination "nom-de-quartier/vill/region/pays" pourra y aller automatiquement juste par l'utilisation des panneaux indicateurs.
3. ○Implémentez les panneaux automatisés version 2. Dans votre monde, il existera des villes ayant des panneaux version 1, et des villes avec des panneaux version 2.
4. ○Implémentez les panneaux automatisés version 3. Encore ici, il existera des villes ayant encore des panneaux version 1 et des villes ayant des panneaux version 2. Et tous vos panneaux devront néanmoins continuer de fonctionner quelque soit la version.
5. ☼Certains pays assez méfiants par rapport à certains de leurs voisins, refusent de communiquer trop d'information quand à leur structure interne et ne font éventuellement confiance qu'aux informations de coût de certains autres pays. Chaque pays possède quelques uns de ces panneaux frontaliers. Développez un type de panneaux frontaliers automatisés permettant tout de même à votre voyageur d'arriver à destination.

Mots-clefs : Réseaux, Routage statique, Routage dynamique, RIP, OSPF, BGP



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

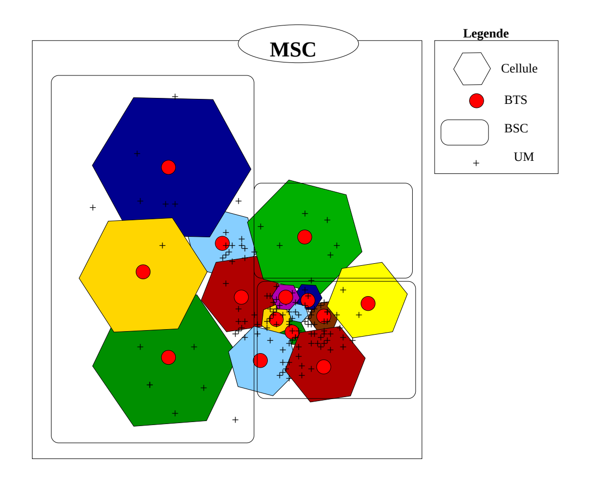
24- Simulation simplifiée d'un réseau GSM**

Le but de ce projet est de simuler la couverture d'un réseau GSM simplifié et l'itinérance des téléphones portables dans ce réseau.

Le réseau GSM est constitué d'un ensemble de stations de base (BTS) sur l'ensemble du territoire que l'on souhaite couvrir, de telle sorte que la station mobile (MS) soit toujours à moins de quelques kilomètres de l'une d'entre elles.

Une cellule, est la surface sur laquelle le téléphone mobile peut établir une liaison avec une station de base BTS. Le principe consiste à diviser une région en un certain nombre de cellules desservies par un relai radioélectrique (la BTS) de faible puissance, émettant à des fréquences différentes de celles utilisées sur les cellules voisines. Ces cellules doivent être contiguës sur la surface couverte. Evidemment, le nombre de fréquences accordées au système GSM étant restreint, l'opérateur est obligé de réutiliser les mêmes fréquences sur des cellules suffisamment éloignées de telle sorte que deux communications utilisant la même fréquence ne se brouillent pas. (Pour info, en France, le GSM opère dans la bande des 900 MHz sur des canaux de 200kHz que se partagent 3 opérateurs).

L'hexagone est la forme régulière qui ressemble le plus au cercle et que l'on peut juxtaposer sans laisser de zones vides. `[...]` Suivant la densité urbaine, le "rayon" de l'hexagone pourra varier de 200m (rue très passante d'agglomération) à plusieurs dizaine de kilomètres (en rase campagne).



La mobilité des abonnés dans un réseau cellulaire a deux conséquences :

- Pour établir une communication, il faut savoir dans quelle cellule l'abonné se trouve. C'est la fonction de gestion de localisation.
- Il doit y avoir continuité de la communication lorsque l'abonné passe d'une cellule à une autre (transfert inter-cellulaire, communément appelé handover).

La bande radio représente la ressource rare et le premier choix architectural fût le découpage du spectre alloué dans un plan temps / fréquence pour obtenir des canaux physiques pouvant supporter une communication téléphonique.

Multiplexage fréquenciel (FDMA) permet de diviser une plage de fréquence en bandes de fréquence.

Multiplexage temporel (TDMA) Pour le GSM, chaque porteuse est divisée en intervalles de temps (IT) appelés slots. A chaque time slot, on associe un nombre connu par la station de base (BS) et le mobile (MS). Le numérotage des slots est cyclique sur une durée définie. L'accès TDMA (Time Division Multiple Access) permet de partager entre différents utilisateurs une bande de fréquence donnée et, sur une même porteuse. Chaque utilisateur utilise alors un slot de la trame TDMA.

Avec C canaux et T intervalles de temps par canal, on a donc un système qui allie un multiplex fréquentiel (FDMA - Frequency Division Multiple Access) et un multiplex temporel (TDMA - Time Division Multiple Access). Un canal physique est donc défini par :

- un numéro de Time Slot TS (dans une trame TDMA).
- une fréquence

Un BSC (Base Station Controller) gère plusieurs BTS. Le MSC (Mobile Switching Centre) interconnecte le réseau GSM avec d'autres réseaux (dont le fixe) et la base de données gérant les abonnés.

- ◖ Genérez une interface permettant à l'utilisateur de placer les BTS (avec leur couverture), les BSC et des usagers de portables. Votre système devra proposer automatiquement les canaux des cellules ainsi formées en respectant les règles d'attribution des canaux sur des cellules contigües. Un nombre (défini par l'utilisateur) d'usagers seront ensuite répartis aléatoirement sur l'ensemble du territoire. Chaque usager sera identifié par un numéro de téléphone (généré par votre système) et enregistré sur le HLR.
- ◖ Créez un mécanisme de simulation des communications entre des paires aléatoires d'usagers. Il devra être possible à l'utilisateur de :
  - paramétrer la probabilité d'appel des utilisateurs d'une zone donnée (zone urbaine, grand évènement sportif, etc.) ainsi que le temps moyen d'une communication.
  - visualiser les trames d'une conversation sélectionnée au sein des multiplex et son acheminement entre les deux interlocuteurs.
  - visualiser à tout moment les états des BSS, BSC, HLR et MS.
- ○/☼Certains MS se déplacent tout en conversant, et passent d'une cellule à l'autre (handover), gérer ce mécanisme au sein de votre réseau.

Références :

`http://www.commentcamarche.net/forum/affich-6940285-cours-en-reseau-gsm-et-gprs`

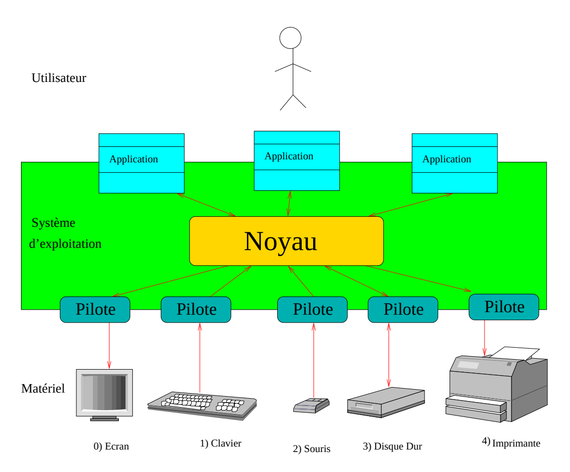
Mots-clefs : Signal, GSM, simulation



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

25- Création d'un simulateur de mini-système d'exploitation**

Le but de ce projet est d'émuler le comportement d'un mini-système d'exploitation, de ses périphériques et de ses applications.



Simulation des périphériques et de ses pilotes

Dans un premier temps, il s'agit de simuler le comportement de périphérique, puis d'implémenter les pilotes (*driver*).

Les pilotes de périphériques (tous définis par un numéro unique) permettent au noyau via des primitives de lecture (*read*), d'écriture (*write*) et de commandes (*ioctl*) d'interagir avec les périphériques qu'ils commandent. Il y a un pilote adapté à chaque périphérique.

Certains périphériques comme le clavier, souris, micro, etc. n'autorisent que la lecture (le système ne peut que lire les signaux envoyés par ces périphériques). D'autres périphériques comme l'écran, l'imprimante, le haut-parleur n'autorisent que l'écriture (le système ne peut qu'envoyer des signaux vers ces périphériques). Enfin d'autres périphériques comme le disque dur permettent à la fois la lecture et l'écriture (le système peut lire ou écrire ce qui est stocké à un emplacement du disque). La commande (ioctl) permettra dans ce dernier cas par exemple de positionner l'emplacement du disque où lire et écrire.

Pour des raisons de simplicité, on n'implémentera pas de systèmes de fichiers, on définira simplement des slots de stockage de taille fixe identifiés par des numéros sur le disque dur.

slot1	slot2	slot3	slot4	...	slotn

Simulation du noyau

Un processus correspond à l'exécution d'un programme. On définira un processus simple par les deux zones suivantes (au lieu de 4 dans un processus réel) :

- Une zone de code contenant les instructions du programme. Sa taille et son contenu ne varient pas au cours de l'exécution.
- Une zone d'allocation statique stocke les variables qui durent tout le temps de l'exécution du processus : les variables globales et les variables locales statiques. Sa taille est fixée, son contenu peut changer.

Le noyau manipule deux types seulement : entier (int) et chaine (string), et offre les fonctions (primitives) suivantes :

- appel des 3 fonctions de la section précédente : read, write et ioctl des pilotes désigné par leur numéro
- Opérations arithmétique : add, sub, mul, div, etc.
- Opérations sur les chaînes : atoi (conversion de chaine vers entier), itoa (conversion de entier vers chaine), comparestring (comparaison de deux chaines), concat (concatenation de deux chaines), etc.
- forkexec : lancement d'un autre processus en parallèle.

Ecriture d'application

Une application (= un programme) sera écrite dans notre système sous forme d'un langage interpreté utilisant les primitives du noyau défini ci-dessus. Il comportera de plus, un ensemble d'instructions permettant la réalisation de tests et de boucles (ou de sauts).

Lors du lancement d'une application, un processus sera crée comportant les instructions données par le langage et exécuté concurrentiellement par le système (qui peut gérer plusieurs processus simultanés).

Exemple d'application (la syntaxe donnée ici vous donne une idée du langage qu'on attend de vous. Vous n'êtes pas obligés d'utiliser exactement cette syntaxe).

```
// Programme "lire_slot_1" qui demande une chaine à l'utilisateur, puis
// stocke la chaine dans le slot 2 du disque dur
string x

ioctl (3, "o2")   // On envoie la commande d'ouverture du slot 2
                  // au périphérique numéro 3 (disque dur)
write (0, "Tapez une phrase :") // Ecrit la chaine "Tapez une phrase" sur
                                // le périphérique 0 (écran)
x = read (1)      // On lit ce qui est dans le périphérique 1 (clavier) et
                  // on stocke la chaîne dans la variable x
write (3, x)      // On stocke le contenu de x dans le slot courant (2) du périphérique numéro 3 (disque dur)
ioctl (3, "c")    // On ferme le slot courant (2) du périphérique 3 (disque dur)


// Programme "lire_slot_2" qui affiche à l'écran, le contenu du slot 2
// du disque dur
String x

ioctl (3, "o2")   // On envoie la commande d'ouverture du slot 2
                  // au périphérique numéro 3 (disque dur)
x = read (3)      // On lit le contenu du slot courant (2) du
                  // périphérique numéro 3 (disque dur)
write (0, x)      // On écrit la chaine x sur le périphérique 0 (écran)
ioctl (3, "c")    // On ferme le slot courant (2) du périphérique 3 (disque dur)


// Programme "calculatrice" qui demande 2 nombres à l'utilisateur, un
// opérateur, puis réalise l'opération et l'affiche à l'écran
string ch1
string o
string ch2
string chres
int op1
int op2
int res

ch1 = read (1)    // On lit ce qui est dans le périphérique 1 (clavier) et
                  // on stocke la chaîne dans la variable x
o = read (1)      // On lit ce qui est dans le périphérique 1 (clavier) et
                  // on stocke la chaîne dans la variable y
ch2 = read (1)    // On lit ce qui est dans le périphérique 1 (clavier) et
                  // on stocke la chaîne dans la variable y
op1 = atoi (ch1)  // Appel de fonction de conversion de chaine en entier
op2 = atoi (ch2)  // Appel de fonction de conversion de chaine en entier

if comparestring (o, "+")
then res = add (op1, op2)
elif comparestring (o, "-")
then res = sub (op1, op2)
elif comparestring (o, "*")
then res = mul (op1, op2)
elif comparestring (o, "/")
then res = div (op1, op2)
else
write (0, "erreur")
exit

chres = itoa (res)
write (0, chres)


// Programme "editeur de texte" : on demande à l'utilisateur dans quel
// slot il veut stocker le texte, puis on stocke tous ce qui est tapé au
// clavier par l'utilisateur jusqu'à ce que l'utilisateur tape EOF
string ch
string slot

write (0, "Dans quel slot stocker le texte ?")
slot = read (1)
slot = concat ("o", slot)
ioctl (3, slot)

prog :
  ch = read (1)
  if comparestring (ch, "EOF")
     goto fin
  else
     write (3, ch)
     goto prog

fin :
     ioctl (3, "c")
```

Concurrence et ordonnancement Le système est multitâche, c'est à dire qu'il donne l'illusion de traiter plusieurs processus "en même temps". C'est à dire que le CPU exécute un certain nombre d'instructions d'un processus, puis fige cet état pour ce processus et passe à l'exécution des processus suivant, enfin, il revient sur le processus exécuter un certain nombre d'instruction à l'endroit où il s'était arrêté, etc.

Le but du programme à réaliser est de :

- Simuler les différents périphériques, dont au minimum :

  - un clavier
  - un écran
  - plusieurs disques durs (chacun ayant un numéro de périphérique différent)
- ◐ Implémenter les pilotes correspondants.
- ◐ Implémenter les différentes primitives du noyau et une petite bibliothèque de fonctions.
- ○Implémenter les processus et leur exécution à partir des instructions interprêtés
- ☼Implémenter la gestion concurrente des processus
- ☼Ecrivez quelques applications que vous lancerez par `forkexec (slot_dans_lequel_se_trouve_le_programme)`.

Mots-clefs : Assembleur, Système d'exploitation, Programmation système



T.T. Dang Ngoc
dntt@u-cergy.fr

26- Création de simulateurs des éléments algorithmiques de bases*

Afin de permettre à des utilisateurs d'appréhender les éléments algorithmiques de base que sont :

- les graphes (orientés, non-orientés, avec ou sans poids) ;
- les arbres (n-aire, équilibrés ou non), tas ;
- les listes, pile, files, tableau ;
- les tables de hachages.

il serait intéressant de fournir une interface permettant de visualiser ces différentes structures et de les manipuler intuitivement :

- initialisation des structures ;
- ajout, suppression, parcours : suivant le type de structure ;
- algos de tris courants sur les structures ;
- opérations spécifiques aux structures ;

L'accent est mis sur le côté pédagogique de l'interface, où l'utilisateur doit visualiser graphiquement les structures et suivre graphiquement le déroulement des opérations qui s'y effectuent (pas à pas). Ainsi pour le tri d'un tableau par tri bulle par exemple, les échanges des éléments, et la limite du tableau des éléments déjà triés doivent être représentés visuellement et permettre à l'utilisateur de suivre intuitivement le déroulement de l'algorithme.

L'utilisateur doit pouvoir paramétrer autant que possible ses structures et ses opérations.

Des statistiques pertinentes (temps de réponses, nombre d'échanges effectués, ...) doivent également être fournies à l'utilisateur.

L'interface doit être aussi intuitive que possible et permettre à l'utilisateur d'interagir autant qu'il le veut (déroulement pas à pas, retour en arrière, exécution continue, arrêt, etc.)

Mots-clefs : Algorithme, pile, file, arbre, graphe, tris, tas, tables de hachage, tableau



Licence 2 I
2010–2011
Génie logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

Conditions générales sur le projet

1 Travail à effectuer

Vous réaliserez le programme demandé en Java.

Chaque prototype devra comporter deux types d'exécution :
- une exécution (pour du batch ou du débogage) en mode console
- une exécution avec une interface conviviale pour l'utilisateur final pour utiliser votre projet

Un rapport en LaTeX(OBLIGATOIREMENT) d'une vingtaine de pages devra également être fourni avec votre projet. Il devra inclure :
- la présentation du sujet et son analyse ;
- la conception de programme, son architecture. Vous expliquerez soigneusement les choix que vous avez faits. Vous parlerez aussi des alternatives qui se sont offertes à vous et pourquoi vous avez opté pour tel ou tel choix de conception ;
- le déroulement de l'implémentation : quels outils vous avez ou n'avez pas utilisés et pourquoi ? Quelles classes java avez-vous utilisées ? Quelles ont été les difficultés rencontrées ;
- des exemples de scénarios ou de tests (éventuellement à intégrer dans votre programme) ;
- l'utilisation de votre programme : c'est-à-dire le mode d'emploi ou manuel d'utilisateur. Vous devrez décrire comment lancer le programme, quelles sont les commandes (et arguments) à taper durant la session interactive ;
- vos remarques : quelles fonctionnalités rajouteriez-vous et comment, quelles ont été les différentes alternatives, comment s'est effectué le découpage de votre programme, de la conception ?
- la gestion du projet : quelle a été la plannification de vos tâches (diagramme de Gantt), comment se sont réparties les tâches entre vous ?
- vos références et bibliographie : il se peut que vous ayez eu la curiosité de chercher dans des livres ou sur l'Internet des documents se rapportant à votre sujet. Loin d'être pénalisant, cela montre votre capacité
  - à vous documenter sur le sujet et ses applications,
  - à analyser la manière dont les logiciels proches de votre sujet pourraient être adaptés à votre problème
  - à comprendre en quoi les algorithmes existants peuvent répondre à votre problème
  - ou vous donner des indices pour y répondre.

Il est inutile de mettre les sources de votre programme en annexe de votre rapport.

2 Attention

Pour éviter tout malentendu, veuillez noter les avertissements suivants :
- Même si vous trouvez par bonheur un logiciel avec ses sources qui répondent exactement au sujet que vous aurez choisi, une simple copie du logiciel ne suffira pas, puisque lors de la soutenance et dans le rapport, il faudra :
  - pouvoir expliquer exactement les algorithmes utilisés et pourquoi ils sont écrits de cette manière ;
  - d'expliquer les différentes étapes de la conception et l'architecture ;
  - pouvoir expliquer en détail n'importe quel morceau de code.
- chaque membre du binôme pourra recevoir des notes différentes si lors de la soutenance et lors du présentiel, la différence d'investissement de chacun se voit de manière flagrante.

3 Recommandations

Vous serez notés sur les points suivants (par ordre décroissant d'importance) :

1. Le programme fonctionne-t-il ? *Il vaut mieux un programme qui ne fait pas trop de choses mais qui marche bien qu'un programme qui fait beaucoup de choses mais qui ne marche pas.*

2. Le programme est-il correct ? *Ce n'est pas parce qu'un programme marche qu'il est correct.* La conception des classes et des paquetages est-elle bonne ?
3. La clarté du rapport : la conception du programme est-elle bien expliquée ? Ne négligez pas le rapport : non seulement il compte pour une part significative de votre note, mais il permet aussi au correcteur de comprendre votre code (ou ce que vous avez *voulu* faire) si celui-ci est mal écrit ainsi que de juger la conception de votre programme.
4. Le soin apporté à l'implémentation : propreté du code (indentation, **convention de codage**[1])
5. Pourvu que votre programme fonctionne, un bonus vous sera accordé pour d'éventuelles améliorations significative de votre programme (stockage et chargement des données dans un fichier, interface graphiques, fichiers de configuration, etc.)
6. La soutenance avec démonstration de votre programme : présentation claire et succinte du sujet, de l'architecture du code en général, et démonstration de votre logiciel telle que vous le feriez à un client.

Organisez votre répertoire de façon logique : src, rapport, ...

4 Modalités de remise du projet

La date limite de remise des projets est fixée au __ **mai 2011**, à **17h** (heure de Paris ;-). La date étant décidée durant le cours, et le sujet donné suffisamment à l'avance, il n'y aura AUCUNE possibilité de report de dates. Tout retard se répercutera sur la note finale.

Le projet doit se faire en **Java** avec les classes standards fournies par le jdk avec les bibliothèques standards. Le projet se fait en **binôme** (deux personnes).

Avant cette date, vous devrez déposer votre projet (rapport compris) sur la plateforme `moodle` à l'adresse suivante :

`http ://moodle.u-cergy.fr`

Un seul projet par binôme devra être déposé.

Quelques éléments importants :
– le répertoire de travail doit être nommé `$HOME/PROJET-GL-`*nom1-nom2* [2] (et tous ses sous-répertoires) Vous y mettrez tout ce qui constitue votre projet (documentation, sources, jeux de tests, `Makefile` ou scripts éventuels, etc.) ;
– un fichier `A_LIRE` doit indiquer au correcteur comment compiler vos classes (quel est le script ou la commande à lancer, quel CLASSPATH positionner, quel fichier de configuration éditer...) et comment lancer le programme ;
– indiquez le prénom et le nom de chacun des binômes dans un fichier AUTEURS ;
– effacez tous les fichiers .class, fichier PS ou PDF généré, core, etc. C'est-à-dire tout ce qui peut être regénéré depuis vos sources ;
– archivez et compressez votre répertoire projet en un seul fichier en tapant :
`tar cvfz projet-gl-nom1-nom2.tgz $HOME/PROJET-GL-`*nom1-nom2*
Le fichier `projet-gl-`*nom1-nom2*`.tgz` généré sera le fichier à déposer avant la date limite.

Si vous avez la moindre question concernant ce projet, envoyer un mail à :
`Marc.Lemaire@u-cergy.fr et Tianxiao.Liu@u-cergy.fr`, `dntt@u-cergy.fr` ou `Mai.Nguyen-Verger@u-cergy.fr`.

[1]en particulier les majuscules minuscules dans le nom des classes : *ExempleDeClasse*, des noms de paquetages : *projet.loisir*, des noms de méthodes : *inscrire, inscrireAdherent*, des variables : *maValeur, valeur* et des variables constantes *MA_CONSTANTE*. Toutes ces conventions sont expliquées en français à l'adresse suivante : `http ://cyberzoide.developpez.com/java/javastyle/` : commentaires appropriés, nom des méthodes, variables, classes et paquetages.

[2]En remplacant `nom1-nom2` par les noms de chaque membre du binome, ne mettez ni accents ni d'espace.